Vesa Nikkilä

Intelligent Wear Plate Condition Monitoring System

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Tampere, March 19, 2015

Thesis supervisor:

D.Sc. (Tech.) Ilkka Seilonen

Thesis advisor:

D.Sc. (Tech.) Mika Karaila

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Vesa Nikkilä

Title: Intelligent Wear Plate Condition Monitoring System

Date: March 19, 2015 Language: English Number of pages: 9+67

Department of Automation and Systems Technology

Professorship: Information and Computer Systems in Automation Code: AS-116

Supervisor: D.Sc. (Tech.) Ilkka Seilonen

Advisor: D.Sc. (Tech.) Mika Karaila

Wear plates are a necessity in quarrying and in mining industry. Wear parts are needed in locations where abrasion is present. Abrasion is caused by the flowing stone material on the surface of the plate. Heavy duty abrasion resistant plates are required in such positions to protect the industrial machinery from damage. The material typically comprises different size stones and they cause wear plates to vibrate for a short period of time when contacting the surface of the plate. This vibration has a specific frequency which changes during the lifetime of the wear plate. In the beginning of the wear plate life cycle, the natural frequency is high, but the frequency slowly drops as the material flow consumes the plate making it thinner.

Currently wear plate monitoring is manual process and the supervision of the plates is time consuming. This work studies the possibility to automate the wear plate condition monitoring basing the analysis on the frequency analysis of the wear plate vibration. The thesis provides a prototype of a monitoring system. Work examines the applicability of two different wireless communication protocols, Bluetooth Low Energy and IEEE 802.15.4, to use with the vibration sensor. Vibration is measured with high-speed MEMS accelerometer. Measurements are passed through gateway to a cloud service for storing and analysis. Although this work only concentrates on wear plates, the idea could be exploited with other wear parts in quarrying and mining.

This work is done for Metso Automation. The project is a part of the Arrowhead project, which is funded by the European Union.

Keywords: IoT, WSN, BLE, 802.15.4, Node.js, ROA, REST, FFT

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Vesa Nikkilä

Työn nimi: Älykäs kulutuslevyjen kunnonseurantajärjestelmä

Päivämäärä: 19. maaliskuuta 2015 Kieli: Englanti Sivumäärä: 9+67

Automaatio- ja systeemitekniikan laitos

Professuuri: Automaation tietotekniikka Koodi: AS-116

Valvoja: TkT Ilkka Seilonen

Ohjaaja: TkT Mika Karaila

Kulutuslevyt ovat välttämättömiä malmin louhinnassa ja muussa kaivostoiminnassa. Kulutuslevyjä käytetään kohteissa, joissa esiintyy louhintamateriaalista aiheutuvaa kulumista eli abraasiota. Yleisimmin tämä materiaali koostuu erisuuruisista kivistä, ja koneen rakennetta suojaavien kulutuslevyjen käyttö on tarpeellista kohteissa, jotka ovat kosketuksissa louhitun materiaalin kanssa. Virtaavan kiviaineksen yksittäisen kiven osuma kulutuslevyyn saa levyn värähtelemään. Värähtelystä on erotettavissa suuruusluokaltaan poikkeava, muista erottuva taajuus, jota kutsutaan levyn ominaistaajuudeksi. Tämä taajuus muuttuu levyn eliniän myötä.

Nykyinen kulutuslevyjen kunnonvalvonta perustuu ihmisen tekemiin havaintoihin sekä arvioihin. Tässä työssä tutkitaan mahdollisuutta automatisoida kulutuslevyjen kunnonvalvontatoiminto perustuen levyjen ominaistaajuuksien muutokseen. Työssä arvioidaan ja käytetään hyväksi langattomia teknologioita, joita tarvitaan siirettäessä suoritetut värähtelymittaukset kulutuslevyltä eteenpäin. Erityisesti työssä tutkitaan Bluetooth Low Energyä ja IEEE:n 802.15.4 standardiin perustuvaa liikennöintiä. Kulutuslevyn värähtelyä mitataan MEMS-kiihtyvyysanturilla. Työ oleellisesti keskittyy vain kulutuslevyihin, mutta värähtelyyn perustuvan kunnonvalvonnan laajentamista myös muiden kulutusosien kunnonvalvontaan mietitään.

Tämä työ on tehty Metso Automaatiolle osana Euroopan Unionin rahoittamaa Arrowhead projektia.

Avainsanat: IoT, WSN, BLE, 802.15.4, Node.js, ROA, REST, FFT

Preface

Kiitän ohjaajaani TkT Mika Karailaa, kuka tarjosi minulle mahdollisuuden tehdä diplomityöni Metso Automationille. Koin työn aiheen tosi mielenkiintoiseksi, ja olen oppinut työtä tehdessäni valtavasti langattomien vähäenergisten verkkojen protokollista. Kiitän myös valvojaani TkT Ilkka Seilosta tärkeästä tuesta työn aikana sekä työn kommentoinnista. Kiitos VTT:lle työn pohjustuksesta.

Ison kiitoksen annan Joakim Gebartille (Eistec AB), kenen ansiosta pääsin nopeasti käyttämään Eistecin 802.5.14 standardiin perustuvaa sensorialustaa. Lisäksi kiitos kuuluu kaikille kollegoilleni: Yiqingille, Teemulle, Laurille, Esalle ja muille.

Kiitos perheelleni ja kavereilleni. Kaippa se on sitten suunnilleen tätä myöten taputeltu.

Eistec AB, `http://www.eistec.se/`

Tampere, March 19, 2015

Vesa Nikkilä

Contents

Glossary

AFH Adaptive Frequency Hopping. 15

API Application Programming Interface. 37, 41

ATT Attribute Protocol. 17, 19, 48

BLE Bluetooth Low Energy. 2, 14, 15, 17, 19, 20, 27, 37, 38, 40, 41, 43, 50, 52–54, 59, 60

CAP Contention Access Period. 23

CCA Clear Channel Assesment. 24

CFP Contention Free Period. 23

CoAP Constrained Application Protocol. 27–31, 34, 37, 40, 43, 44, 46, 51–54, 59

CoRE Constrained RESTful Environment. 33, 51

CRC Cyclic Redundancy Check. 16

CSMA-CA Carrier Sense Multiple Access with Collision Avoidance. 23, 24

DFT Discrete Fourier Transform. 10, 45

DIRT Data-intensive real-time. 36

DLL Data Link Layer. 25

FCS Frame Check Sequence. 26

FFD Full Function Device. 21, 22

FFT Fast Fourier Transform. 38

FIFO First In-First Out. 13, 38

GAP General Access Profile. 17, 20

GATT Generic Attribute Profile. 17, 19, 20, 47, 52

GFSK Gaussian Frequency Shift Keying. 16

GTS Guaranteed Time Slot. 23

HCI Host Controller Interface. 15, 16

HPF High-pass filter. 14

HTTP Hypertext Transfer Protocol. 14, 26, 28–30, 32–35, 37, 38, 40, 41, 43, 46, 52, 53, 60

I2C Inter-Integrated Circuit. 12

IEEE Institute of Electrical and Electronics Engineers. 21

IoT Internet of Things. 59, 61

L2CAP Logical Link Control and Adaptation Protocol. 17, 20

LR-WPAN Low-Rate Wireless Personal Area Network. 21

M2M Machine-To-Machine. 28

MAC Medium Access Control. 25, 26

MCPS MAC Common Part Layer. 25, 26

MCU Microcontroller Unit. 38, 43

MEMS Microelectromechanical systems. 12, 14

MFR MAC Footer. 26

MHR MAC Header. 26

MLME MAC Layer Management Entity. 23, 25, 26

MPDU MAC Protocol Data Unit. 25, 26

MSE Mean Squared Error. 45

MTU Maximum Transmission Unit. 19, 20, 50

ODR Output Data Rate. 12, 13, 38

OLS Ordinary Least Squares. 11

OSI Open Systems Interconnection. 21, 25

P2P Peer-to-Peer. 17, 21, 22

PAN Personal Area Network. 21–23, 26, 37, 38, 44

PDU Protocol Data Unit. 18, 19, 50

PHR PHY Header. 25

PIB Personal area network information base. 24, 26

PLME Physical Layer Management Entity. 24

PPDU PHY protocol data unit. 24, 25

PSDU PHY service data unit. 25, 26

REST Representational state transfer. 31–34, 41, 52, 53

RFD Reduced Function Device. 21, 22

ROA Resource-oriented Architecture. 32–35, 40

SAP Service Access Point. 22, 24, 26

SHR Synchronization Header. 25

SLIP Serial Line Internet Protocol. 44

SMP Security Manager Protocol. 17, 20

SPI Serial Peripheral Interface. 12, 38

SUID Sensor Unique Identifier. 33

TCP Transmission Control Protocol. 26, 29, 45

UDP User Datagram Protocol. 29

URI Uniform Resource Identifier. 26, 27, 32–35, 51, 53, 54, 63

UUID Universally Unique Identifier. 20

WSN Wireless Sensor Network. 14, 38

1 Introduction

1.1 Motivation

A wear plate is a wearing part that is used with various mining and quarrying equipment, including crushers, buffer bins, conveyors, dump trucks and front loaders. A wear plate is typically made from abrasion resistant steel, commonly Hardox 500, but some applications can take advantage of ceramic, rubber. It is common to combine features of these materials to reduce noise and increase the lifetime of the plate.

The wear plate is abraded when the processed material is in contact with the surface of the plate. In most cases, the material is comprised of different size rocks. Wearing occurs when a stream of stone material flows on the surface of the plate slowly consuming a wide valley to the plate. The abrasion will make the plate thinner. A thin plate is prone to breaking and it must be replaced with a new one during the downtime of the process. Depending on the application and the level of wearing, the lifetime of a single wear plate can vary from several months up to over one year. Some plates may even last several years or longer. Plates which are located in positions with less abrasion also last longer. Machine can be damaged if a worn wear plate breaks. Size and dimensions of a wear plate is determined by its application. There is countless amount of different wear plates for different applications.

When a stone hits the plate, it causes the plate to vibrate briefly on certain frequencies. One of these frequencies is dominating and it is known as the natural frequency of the plate. Each different wear plate has its characteristic natural frequency which decreases as the plate becomes thinner. If the change of this frequency is known for a certain wear plate during its lifetime, it is possible to estimate the approximate remaining lifetime from current frequency. There are also other viable methods to estimate the thickness of the plate, e.g., it is possible to measure the average amplitude of the vibration. Thin plate is more flexible; the vibration has higher amplitude.

Metso offers a wide range of supervision and technical support [1]. However, condition monitoring of wear plates is still a manual process and it is based on human perception and estimations. Constant monitoring of wear plates is needed to determine the condition of plates. There are no predefined systematic routines for wear plate conditioning monitoring since it is based on human evaluation. In practice, an employee designated for the condition monitoring performs a cursory inspection of wear plates and makes an estimate when certain plate needs to be replaced. Some wear plates are visible and they can be easily evaluated. Some machines have observation hatches and it is possible to examine the condition of such wear plates with moderate effort. However, condition monitoring can be nearly impossible in some locations. Buffer bins may be filled with incoming stone material and it is not possible to inspect plates without emptying the whole bin. Sometimes it may be hazardous for health to perform monitoring checks. For example, certain locations do not have proper ventilation and the air in the room could be filled with

harmful stone dust.

The benefits of automated condition monitoring are regular monitoring and up-to-date knowledge without risking the health. However, vibration based condition monitoring does not come without challenges. The abrasion of the wear plate may be very localized. Stream of stones often flows through a certain route on the surface of a plate, which wears a valley on the plate. The valley may appear near attachments of the plate, which could make it difficult to determine the abrasion from the natural frequency.

Present supervision and maintenance service business does not include automated condition monitoring. Integrating automated condition monitoring system to the supervision and maintenance service enhances the service quality and increases customer satisfaction. Moreover, suggested service strengthens customer relationship. Data gathered from wear plates on customer sites provides valuable information for further development of the monitoring service. Advanced wear plate monitoring reduces machine downtimes and improves productivity.

1.2 Research objectives

The objective of this work is to automate the wear plate condition monitoring by designing and implementing a prototype of a monitoring system which is composed of a sensor network, a gateway and a cloud. The cloud framework is already implemented and it offers a platform to develop analysis services for sensor data.

First, wired connection betweeen the sensor and the gateway was considered but quick reasoning lead to the conclusion that wired transmission is not a viable option since many locations are very difficult to access with a cable. Noticeable advantage of a wireless sensor is its flexible installation capabilities. For example, wired sensor cannot be installed to spinning applications which would cause the wire to tangle around the axis. Moreover, mounting of the sensor to the plate will be permanent and it is not reusable with replacement wear plate. The work that is required to reroute wires constantly would be time consuming. Therefore, wireless data transfer is cheaper and it is more suitable compared to the wired transfer.

In the designed system, data can optionally be received with a smartphone that supports Bluetooth Core definition 4.0 or higher. Gateway is not necessarily required while smartphone is used but then the user must be in the proximity of sensors that are attached to wear plates. Gateway has a built-in web server that can be accessed from the local network. The web server provides information about the sensor network, e.g., last update time and current natural frequency of each sensor. Gateway forwards the received data to a cloud service for analysis. Natural frequency of the plate is resolved by analyzing the frequency spectrum of the vibration. Results and raw data are stored into a cloud database.

The sensor must remain operational for from several months up to half a year without battery replacement, which restrains the power consumption. Two types of radio protocols are evaluated: the Bluetooth Low Energy and the IEEE 802.15.4 protocol of which both are designed for low power consumption applications.

In addition to the present status of the wear plate, vibration data also provides

information of the expected lifetime of the wear plate. Natural frequencies can be plotted as a function of hours of operation which shows a clear linear relationship between them. With linear fitting it is possible to extrapolate the expected lifetime of the plate. Information of the expected lifetime can be exploited to automatically send right amount of replacement wear plates to the mining site. Moreover, system the will provide the end user with information about costs related to wear plates.

1.3 Research methods

In the beginning of the work, the feasibility of vibration based wear plate condition monitoring was evaluated with sample data (Figure 1.1) gathered by VTT. The gathered frequency data can be estimated with a linear fit. Measurements are lacking from the beginning of the May but still the drop of the natural frequency is clear. In the end of the sample data, the frequency dramatically drops. This is caused by a fracture in the wear plate. The wear plate should have been replaced earlier, since broken plate could cause damage to the machine. For this type of wear plate, natural frequencies over 900 Hz could be considered safe.

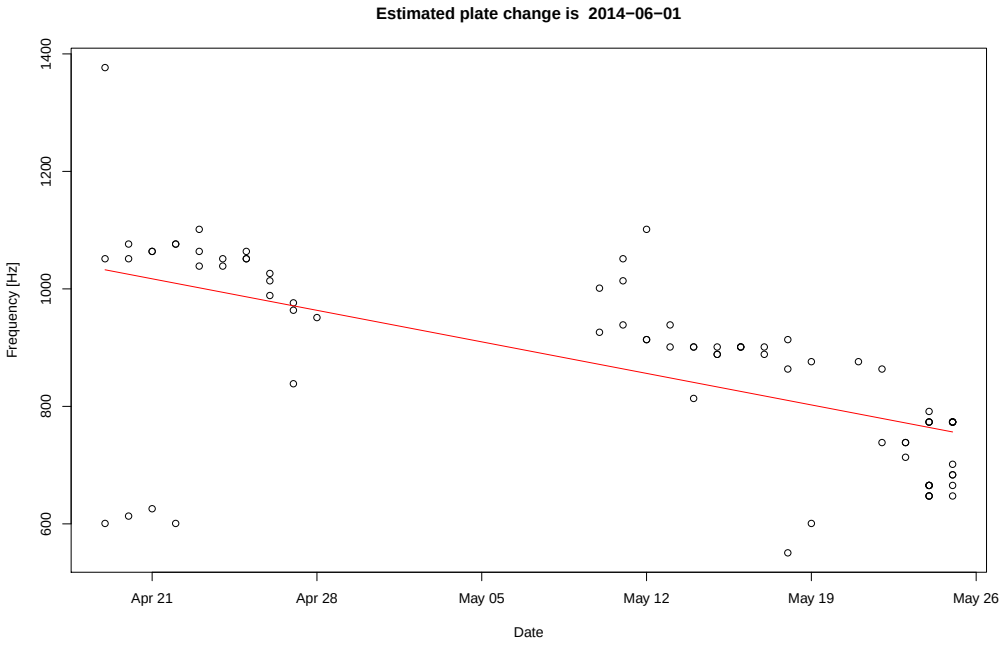

Figure 1.1: Frequency change is linear with respect to the wearing of the wear plate

The main sources for the technological background of this work have been various standards and definitions; especially the Bluetooth Core definition, 802.15.4 standard, and HTTP and CoAP standards. The work began by first designing the hardware of the sensor. The sensor contains an accelerometer, which measures the vibration of the plate. After the data has been collected, it is transmitted wirelessly to the gateway module with Bluetooth Low Energy. The detailed design was

outsourced. While waiting for the sensor hardware, it was possible to simulate the BLE sensor and develop the gateway with Node.js. The gateway is equipped with Bluetooth 4.0 dongle and it is capable of receiving the sensor data. The cloud framework provides Node-RED environment in which it is easy to develop data analysis features. Cloud has an HTTP interface for communication.

In the end of this work, the prototype was tested in real environment. The sensor was installed to a wear plate and measurements were transmitted to the gateway. Measurements were conducted in a chromium mine in Kemi.

1.4 Structure of the work

Section 2 covers the background of the work. Background section explains different components in quarrying. Background also describes the target case to which this work concentrates.

Section 3 tangles with technological side of the work. The section reviews Bluetooth Low Energy and 802.15.4 standard for wireless data transfer. In addition to these two specifications, the section discusses about CoAP and HTTP protocols.

Design and implementation of the monitoring system is presented in the section 4. In the design, the system overview is described and separate system components are connected to each others. Implementation gives detailed description of how the system was actually implemented and what technologies were used in the implementation phase. Different use cases of the system are demonstrated in the section 5. Use cases are divided into three separate sections where all the main system parts (sensor level, gateway level and cloud level) are tested individually with corresponding use cases. After use cases, the section 6 exhibits measurements.

Lastly, section 7 of this work summarizes the work. It reviews the work and evaluates the overall performance. The section discusses further development and expandability of the system. Moreover, section introduces future visions of automated wear plate monitoring and reviews shortly upcoming technologies.

2 Wear parts in quarrying and mining

Before describing the monitoring system, it is good to understand the background of the work and in what kind of environment the monitoring system is planned to be used. This background section explains quarrying process and the necessity of wearing parts in different phases of the process.

2.1 Quarrying and mining

Quarrying is the process of extracting stone, ore or such raw material from the ground [2]. Typically, quarrying takes place on the surface of the ground whereas mining may refer to underground processes. The following text identifies the components of the process, describes it shortly and explains the necessity of wear parts. Moreover, the section classifies different types of wearing that are present in the process.

2.1.1 Components

In order to describe the process of quarrying itself, it is necessary to define the main components of the process and their tasks in it. These components include crushers, feeders, screeners and conveyors [3]. These are the main physical components of the process, but there are also non-physical ones, such as the plant automation systems. Many physical components come in two different versions: stationary and mobile. Stationary devices are capable to process large volumes of material, but they are difficult to move. Mobile stations are easy to move, but on the contrary they cannot typically handle large amounts of material.

Crusher To reduce the size of quarried or mined material (stone, ore, etc.), it must be crushed with a work machine known as a crusher. There are various different types of crushers but on the bottom level they can be divided into two main categories: compressive crushers and impact crushers. A *compressive crusher* is based on compressing the material until it breaks [3]. An *impact crusher* utilizes extreme impacts to cause the stone or ore to fracture. A primary crusher reduces the size of raw material and secondary crushers further reduce the size, if needed.

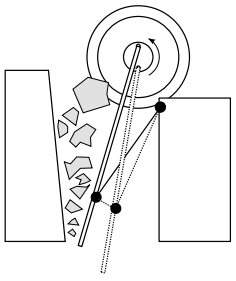

Figure 2.1: Jaw crusher working principle. Derived, simplified [3]

Jaw crushers (Figure 2.1) are common compressive crushers that include a single crushing plate, which compresses the material against the back wall of the machine forcing it to break [3]. Crushing plate is attached to an eccentric shaft that is rotating. Cone and gyratory crushers both have an oscillating shaft and their principle of function resembles the jaw crusher. The crushing takes place in a space called crushing cavity inside the crusher. However, unlike with jaw crushers, there is no crushing plate but instead a cylindrical part which circles in the crushing cavity and compresses the material against walls. Gyratory crushers are often used as primary crushers in high capacity plants. Jaw crusher is also used as a primary crusher when not so high capacities are needed.

Impact crushers break the material with a high intensity impact. Impact crushers offer a high reduction rate and result in a cube shaped product. Conventionally, impact crushers are the last part of the crushing process where the desired result is fine and cube-shaped. Impact crusher may be utilized in primary crushing in applications with low level of abrasion [3].

Feeder Usually quarried material cannot be directly fed to the crusher. Stone material is carried into a feeder with a front loader. Feeder then guides the material flow to a conveyor or to a crusher. Feeders ultimately control the feed rate of crushers. There are two different feeder types.

Apron feeder is a type of feeder that can be used in any kind of quarry application associated with dry, wet or sticky materials. The apron feeder contains a conveyor which conveys and feeds the quarried material to the crusher.

Vibrating feeders are other type of feeders that are designed for large sized material. Vibrating feeders include sections to filter out the fine material and these feeders are mainly used with primary crushers. The conveying of the material is established with vibration rather than conveying belt. Vibrating feeder purchase costs are typically lower compared to apron feeders [3].

Screener Especially in the beginning of the crushing process, the size of quarried material varies and it is desirable to separate it to different size categories. Screener is a device that is able to separate smaller and bigger material. The screener contains a feed box where the material is initially loaded. The material is passed to the screening surface, which contains certain size holes. These holes allow smaller pieces of the material to screen out from the flow. The separation probability is increased with applying vibration to the screener due to the stratification. In stratification, smaller size chunks find their way to the bottom of the flow. The vibration is also responsible of conveying the material on the screen surface.

Conveyor A conveyor is needed in the quarry process to link two process parts together. For example, conveyor is commonly placed between two crushing stations. A typical conveyor utilizes a conveying belt to convey the material. In addition to linking purposes, the conveyor may be used to pile processed material in the end of the process.

2.1.2 Overview of quarry and mining processes

Figure 2.2 presents a refinement process which is common in mining and quarrying [3]. Main activities of these processes are: drilling, blasting, boulder handling, crushing, screening, material loading and hauling of the material. This section only describes the process on a general level. The first phase of the process is obtaining the material either by drilling or by blasting it. The material is loaded into a dump truck with front loaders. Dump trucks haul the material to a feeder which feeds the material either to a screener or straight to a crusher. Screener discards small pieces of the material in order to reduce the unnecessary load of the primary crusher. The produce of primary crusher is later joined with bypassed material flow. If required, this flow is carried to a secondary crusher to further reduce the size. The operation is repeated until required material size and shape is achieved.

In the final stage of the process, the product is piled, loaded, e.g., into a dump truck and delivered to a factory which uses the product as a raw material.

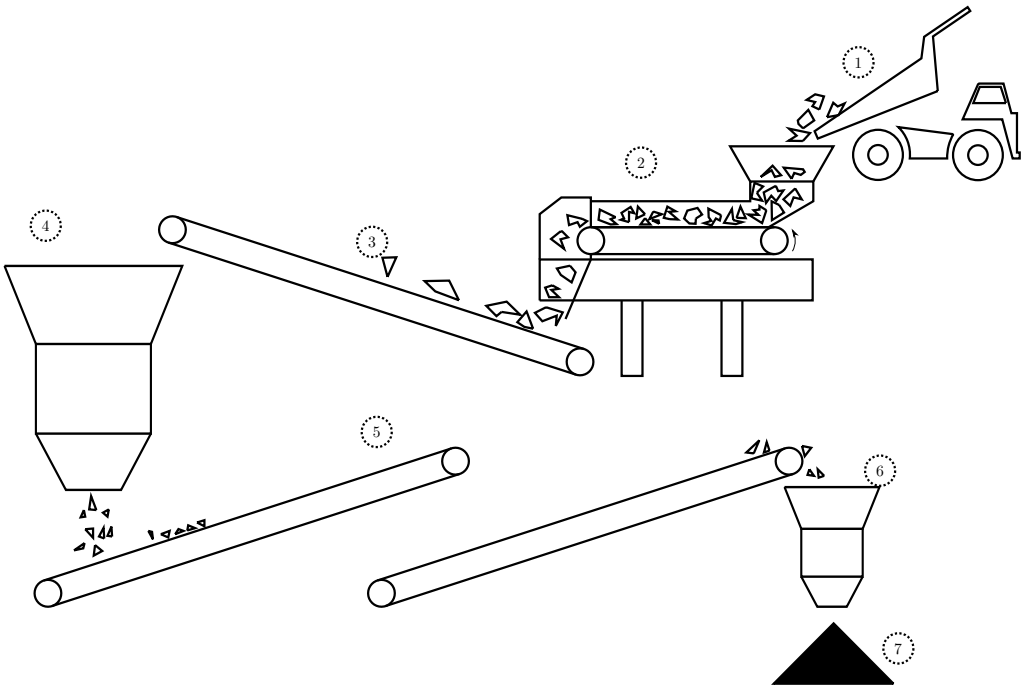

Figure 2.2: Example process. Dump truck (1) dumps the quarried material into a feeder (2). Conveyor (3) conveys the material to primary crusher (4). Conveyors (5) convey the precrushed material to secondary crusher (6). Secondary crusher produces the final product (7). [3]

Depending on the requirements, the process may utilize either stationary, mobile components or both for optimized efficiency. Stationary components allow processing of greater volumes but they are difficult to move and are not practical in temporal processes or with processes that need to change location from time to time. Mobile stations are easy to move and they can completely eliminate the need of separate

haulers. Hauling forms the biggest share of costs in stationary processes with 28%. For mobile processes, the same share is reduced to 11% [3]. Wearing parts form approximately 10% of the total costs on average in both cases [3]. Other costs of the process include blasting, crushing and screening. When minimizing the total cost, it is important to remember that reducing certain cost may increase other costs.

2.1.3 Wearing parts

Mineral processing and stone crushing unavoidably result to wear of certain machine parts. Wearing parts are directly in contact with the processed material. Wearing can be divided into following three types: compression, impaction and sliding [4]. These three types of wearing are the main types of wearing. Compressive wearing occurs when material is being squeezed in between of two wear parts. Compressive wearing is common for jaw crusher which utilizes compression to break stones. Material hitting fast to a wear part causes impactive wear. Such wearing is typical for impact crushers. Sliding wear is caused by material flowing freely on a wear surface. Sliding wear is typical for feeders and screeners.

2.2 Target case

Wear plate vibration measurement of this work are conducted in chromium mine in Kemi, Finland. The wear plates are located in a nozzle that feeds the conveyor. The feed receives the pre-crushed ore and directs it below to the conveyor. Inner surface of the feed is covered with wear plates to dampen the impact of the falling ore, which is dropped from certain altitude.

The Figure 2.3 shows a simplified view of the process. First, the ore is dropped into the nozzle. The falling ore then hits the wear plate on the inner side of the nozzle. The impact of the ore causes the wear plate to vibrate. A sensor is attached on the back side of the wear plate. The sensor wakes up and starts measuring instantly after receiving the inertial interrupt that is above specified threshold level. The measurement lasts only fraction of a second but it is adequate time to deduce dominating frequency components of the vibration. In the described target case, wear plates typically have dominating frequency component approximately in 1000 Hz. This frequency is called as the *natural frequency* of the plate.

The ore bounces down and lands on the conveyor after hitting the wear plate on the side of the feed nozzle. The chromium ore is then conveyed for secondary crushing and for enrichment process. However, it is not important to inspect the process further in the scope of this work.

Wearing of a wear plate The ore slides and bounces down the wear plate to the conveyor as the Figure 2.3 depicts. The ore always slides down approximately the same route which locally abrades the wear plate. Eventually the abrasion of the ore stream forms a valley on the surface of the plate like the Figure 2.4 demonstrates. The forming of the valley will speed up the abrasion since the ore will specifically gather into the valley.

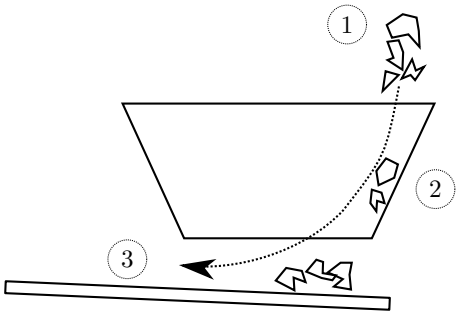

Figure 2.3: Target case studies a feed in nozzle of a conveyor. The ore is dropped (1) into the nozzle and it hits the wear plate on the side of the nozzle (2). After hitting the wear plate, the ore bounces down to the conveyor (3)

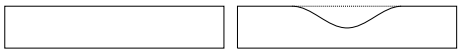

Figure 2.4: Flowing ore consumes the wear plate slowly and forms a valley. New wear plate is shown on the left side and used on the right side

A wear plate with a deep valley is not as stiff as a new plate and thus vibrates with lower frequency. This is the fundamental idea of vibration based wear plate condition monitoring. Originally the idea comes from Erkki Jantunen (VTT).

However, the valley may not always form close to the center of the plate. The location of the valley affects to the drift of the natural frequency. The wear may be difficult to monitor if the valley is near the edge of the plate where it is attached to the work machine.

2.3 Resolving the natural frequency of the wear plate

Sensor measures the acceleration of the wear plate and stores the data in time domain format. In this work, natural frequency is resolved by finding the first peak from the frequency domain presentation. Inspecting the whole spectrum drift rather than inspecting only one frequency spike would guarantee more reliable results. However, this work only utilizes a simple maximum amplitude search for the sake of simplicity. The emphasis of this work is on overall implementation of the system rather than on developing the analysis algorithm.

2.3.1 Frequency domain analysis of the acceleration signal

A good sample of acceleration data measured from the wear plate features a clear damped sinusoidal signal. The simplest method of resolving the natural frequency from such signal is to count the number of oscillations and divide it by the signal duration, which is the sample length N multiplied by the sample rate f_s. Other method of resolving the signal frequency concerns frequency domain analysis. The

time domain signal can be presented in the frequency domain with the Fourier transform [5]. The natural frequency should be visible in the frequency representation of the signal. Collected acceleration signal is not continuous but collected with a sample rate f_s, thus the transform utilizes the Discrete Fourier Transform (DFT). The DFT is defined for finite set of sequential data $\mathbf{x}$ as follows [5, p. 2]:

$$X_k = \sum_{n=0}^{N-1} x_n \exp\left(-i \cdot 2\pi k \frac{n}{N}\right) \tag{1}$$

Each element in the vector $\mathbf{X}$ is a complex number that represents a frequency bin. An absolute value of the element X_i contains information of the magnitude of a certain frequency. Angular position of the transformed point denotes the phase shift of the frequency. The length of the transformation equals to the length of the given signal. Common to the Fourier Transform, the DFT also produces a negative frequency mirror, which is a consequence of sine function properties and half of the transformation can be discarded. The highest resolved frequency is half of the sampling rate which is known as the Nyquist frequency. Because of the mirroring, the center frequency bin covers highest possible frequencies $f_s/2$ and X_0 presents lowest frequencies. Other frequency bins are evenly slotted in between these boundaries.

The resolution of the transform depends on the sample length N and the sample rate f_s (2).

$$\text{resolution} = \frac{f_s}{N} \tag{2}$$

2.3.2 Estimating the remaining lifetime of the wear plate

The analysis of the natural frequency development with respect to the time enables the wear plate lifetime estimation. Change in the natural frequency was analyzed in the beginning of this work. Figure 1.1 shows clear linear correlation between the frequency and time.

Regression concerns relationship between a numeric dependent variable and one or more numeric independent variables [6, p. 160]. A dependent variable is a variable that is to be predicted whereas independent variables are called as predictors, i.e., they are used to estimate the dependent variable. Lines can be defined in a slope-intercept form as follows: $y = \mathrm{a} + \mathrm{b}x$. Here y is the dependent variable and x is the independent variable. Simple linear regression analysis fits the model to the given data so that it relates the supplied x values to y in a best available way. In other words, constants a and b are defined to offer the best possible solution. Common definition for the best estimates of these parameters is the minimum of sum of squared errors [6, p. 164]. This methods is known as the Ordinary Least Squares (OLS). Minimization problem is formulated in the following way:

$$\sum (y_i - \hat{y})^2 = \sum e_i^2 \tag{3}$$

Where y_i presents the y value of ith sample. $\hat{y}$ is the estimate for y that is obtained by defining constants a and b. Thus, $\hat{y}_i$ equals to $\hat{\mathrm{a}}x_i + \hat{b}$. For OLS, it can be showed

that the best estimate for b is [6, p. 165]:

$$b = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{\mathrm{Cov}(x, y)}{\mathrm{Var}(x)} \tag{4}$$

And the optimal value for a is:

$$\mathrm{a} = \bar{y} - \mathrm{b}\bar{x} \tag{5}$$

Correlation indicates how closely the estimate is dependent on the data where it is fitted. Correlation commonly refers to the Pearson's correlation coefficient, which is defined as a ratio of covariance between x and y and product of standard deviations of x and y [6, p. 167]:

$$\rho_{x,y} = \frac{\mathrm{Cov}(x, y)}{\sigma_x \sigma_y} \tag{6}$$

The Pearson's correlation coefficient can have values from -1 to 1. Coefficient values 1 and -1 refer to perfect correlation in which case formed linear estimate has no error. Sign of the correlation coefficient describes the nature of the correlation. If the correlation is negative, then there is a decreasing linear correlation. Zero value and values close to zero signify the lack of any correlation in which case the model does not represent the data, or it represents the data very weakly. Usually, the correlation is thought to be strong if the absolute value of the correlation coefficient is over 0.5.

3 Technology Review

Methods to measure vibration a wear plate and transmit these measurements in practice are studied and compared in this section. This section studies closely Bluetooth Low Energy specification [7] and IEEE 802.15.4 standard [8]. The Hypertext Transfer protocol and the Constrained application protocol are utilized in this work and they are presented in this section. The section introduces comparatively new concept, resource oriented architecture, which dictates how HTTP based applications should be implemented. Finally, this section introduces Node.js environment that is used in the implementation phase of the work.

3.1 Acquiring vibration data from acceleration plates

Data acquisition is a critical part of the monitoring process. Correct estimates cannot be formed without properly measured data. However, the emphasis of this work is on implementing the overall functionality of the monitoring system and providing a prototype rather than on measurements. Still it is good to remember the importance of proper measurements. In this work, the vibration of the plate is measured with an accelerometer. Section 3.1.1 presents high speed digital accelerometer in detail that measures the vibration of the wear plate. Vibration is measured with an accelerometer in this work but it is not the only possibility. Vibration could be also measured, e.g., with a microphone.

3.1.1 Using accelerometer to measure vibration

An accelerometer is a device capable of sensing acceleration that is affected to it. Accelerometers are available as Microelectromechanical systems (MEMS) components. MEMS accelerometers are very small and hence they are easy to integrate to the sensor board. Many accelerometers use Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I2C) bus for transferring measurements to the host or for configuring purposes. Some accelerometers offer analog data outputs. To be able to read an analog value, the host controller needs an analog to digital converter. It is important to select a suitable accelerometer for the purpose. The accelerometer must have an adequate output data rate or the measured signal will be corrupted. Moreover, the amplitude of the acceleration signal needs to be considered. Normally accelerometers reach g-values of 10, but high-g accelerometers can reach over 500g.

LIS3DH accelerometer This work utilizes ultra low-power LIS3DH (ST Microelectronics) digital accelerometer [9]. It is highly configurable and measures three different axes. At maximum speed, the accelerometer measures acceleration with 5 kHz Output Data Rate (ODR). Maximum scale of the accelerometer is 16g. Three different operating modes are provided to control the power consumption: power-down mode, normal mode and low power mode. The accelerometer has better precision in normal mode but it cannot reach as high ODR rates as the low power

mode. The power mode and the measurement frequency is selected with configuration of the first control register. Power consumption of these modes are given in Table 3.1. Power down mode is enabled if 0 Hz output data rate is selected via the control register.

Table 3.1: LIS3DH accelerometer power consumption in different modes and with different ODR values. [9]

ODR [Hz]	**Normal mode** [µA]	**Low power mode** [µA]
0	0.4	0.4
10	4	3
25	6	4
50	11	6
100	20	10
200	38	18
400	73	36
1250	185	Not supported
1600	Not supported	99
5000	Not supported	184

FIFO buffer The accelerometer has a built-in First In-First Out (FIFO) buffer, which is capable of storing 32 samples. New values are stored to the buffer after measuring. Reading output registers frees one slot in the buffer. The value that is read is always the oldest measured value. A single acceleration sample contains data from all different axes. Acceleration values of each axis are stored in two 8-bit registers, high and low; there are six output registers in total. These registers hold the current acceleration information in two's complement format. However, it is possible only to utilize the high register which provides coarse value of the acceleration. The FIFO buffer can operate in four different modes. The operating mode is selected by configuring the fifo control register of the accelerometer.

In the *bypass mode*, the FIFO is not operational. This mode can be used to empty the fifo buffer. *Fifo mode* is the normal operation mode in which the buffer keeps filling continuously. After collecting 32 samples, data collection is stopped and the content of the buffer remains unchanged until it has been reset by toggling the bypass mode. *Stream mode* is similar to the fifo mode: new samples are continuously placed to the buffer at the selected ODR. However, new data is placed to the buffer even after it has been completely filled by overwriting the oldest value. Therefore, the buffer must be read to free slots in the buffer for new samples before overwriting takes place. A strategy where the whole buffer is read in one chunk can be exploited to minimize the host processor load. All slots are freed simultaneously after the read operation is completed. When reading the whole buffer, the process must be faster than 1/ODR minus the start delay read operation or otherwise old values will be overwritten. *Stream-to-fifo mode* is a combination of fifo mode and stream mode. At first, the mode operates in stream mode. The operation mode is switched to the normal fifo mode when an interrupt is triggered. The operation mode returns back to the stream mode after the interrupt has been cleared. The interrupt signal stays

high if the accelerometer is set to latch the interrupt and signal is only set to low after manually reading the interrupt source register.

Other features The accelerometer includes various other features, such as freefall detection, a High-pass filter (HPF) and an internal temperature sensor. The HPF is useful to remove the static acceleration component from the signal caused by the gravity and the feature may be considered to be utilized when calibrating the wear plate sensors.

3.1.2 Other methods

There are also other viable ways of measuring the vibration. One method is to use a MEMS microphone. Microphones can reach higher frequencies than accelerometer. However, microphones are easily affected by the surrounding noise whereas accelerometer only measures the vibration of the plate.

3.2 Data transmission technologies

The data is transferred to the gateway after collecting it with the accelerometer. Here the Bluetooth Low Energy and the 802.15.4 protocols are considered as solution candidates to transfer the data from the sensor to the gateway. Both of these considered protocols are designed for low power consumption applications. Bluetooth Low Energy uses over 1 GHz frequencies with frequency hopping. The 802.15.4 works on sub giga hertz frequency and does not utilize frequency hopping. These two differences make Bluetooth Low Energy (BLE) and 802.15.4 protocols very different from each others on the physical level. Wear plate sensors form a Wireless Sensor Network (WSN). In the sensor network, each device is communicating with the gateway forming a star topology compliant network.

Communication between the gateway and cloud is based on the Hypertext Transfer Protocol (HTTP) protocol. The gateway provides an HTTP server, which the End user can access via a personal computer or a mobile phone. The gateway sends the gathered data to the cloud via the Ethernet for analysis and storing. Traffic between the cloud and the gateway is SSL secured to improve safety.

3.2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless network technology which was first presented in the Bluetooth 4.0 Core definition [7]. It shares many architectural components with the regular Bluetooth but the working principle is completely different. The aim of the design is to improve energy efficiency and lower the cost of the existing Bluetooth definition [10]. Bluetooth Low Energy is a robust wireless low data rate technology and it is suitable, e.g., for different kind of sensors [11]. Bluetooth Low Energy is currently one of the leading candidates for short-range control and monitoring applications [12]. BLE uses 40 different channels of which three channels are used for advertising. Devices can broadcast advertisement data on advertisement

channels. Advertisements are used to promote the peripheral to make it visible for other devices. Advertisement broadcasts can contain information about the status of the device. These broadcasts are kept short for low energy consumption and only 31 bytes are reserved for the actual payload in the advertisement packet.

Channels operate on already crowded 2,408 – 2,48 GHz range. Many of BLE channels overlap with 802.11 WiFi channels, but Adaptive Frequency Hopping (AFH) allows coexistence of these networks [13, p. 8]. Channels that have often interference will be blacklisted and thereby they are not used. Moreover, three advertisement channels are strategically distributed to positions of the frequency spectrum where they do not overlap with WiFi frequencies. Bluetooth Low energy stack is divided into the Host and the Controller. The Host and the Controller communicate through the Host Controller Interface (HCI). Profiles, that define the actual functionality of a BLE device, are implemented on the Host. The whole Bluetooth Low Energy architecture is depicted in Appendix B. Application Layer is on top of the Host and it is the highest layer. Application layer utilizes profiles.

The Controller The Controller forms the lower level of the Bluetooth Low Energy technology [13, p. 47]. It includes the low level design, such as the used modulation and frequencies (Physical Layer). The Controller communicates with other devices with the radio. The Link Layer is part of the Controller and it is responsible for maintaining connections.

Link Layer is a complex part of the BLE architecture. It is responsible of advertising, scanning and creating connections. Moreover, the Link Layer manages established connections. The Link Layer also ensures that packets are not malformed. Link Layer channels are divided into 3 advertising channels and 37 data channels. Data can be transmitted on advertisement channels without establishing a connection to the device. In general, these advertisement channels are for device advertising. The data channels are used after establishing connection to the device. These channels utilize the AFH, which improves transmission robustness. Data is sent via data channels to other device, which optionally acknowledges packets and if no acknowledgement message is received, the packet can be retransmitted.

The Link Layer has an internal state machine, which has the following states: Standby, Advertising, Scanning, Initiating and Connection. The *Standby* state is an idle state, where nothing happens but the device is ready to start advertising, scanning or to initiate a connection. In the *Advertising* state, the Link Layer is allowed to transmit advertising packets and it can also respond to scan requests from other devices by sending a scan response. The *Scanning* enables the device to receive advertising packets on advertising channels. This state is used to discover nearby advertising devices. However, it is not possible to directly connect to another device from the Scanning state. The device must first enter the *Initiating* state via Standby state. In the Initiating state, the receiver listens to the device to which is to be connected. If an advertisement packet is received, the Link Layer sends a connect request to the advertiser and moves into the *Connection* state. The advertising device will as well move to the Connection state. The Connection state

has two substates that are different for a master and a slave device. The device enters the *Master substate* if it is the initiator of the connection. This substate can only be entered from the Initiating state. Advertising device enters the *Slave substate* from the Advertising state.

A small packet structure is defined for sending data on any data or advertising channel. This packet is called as the Link Layer packet and the structure is depicted in Figure 3.1.

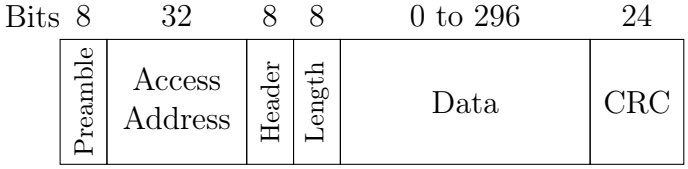

Figure 3.1: The Link Layer packet structure. Derived [13]

In this packet format, 8-bit Preamble synchronizes the receiver bit timing and sets the automatic gain control. The Access Address is fixed with advertisement packets but it is random with data packets. The Header describes the content of the packet and the Length defines the length of the payload (Data field). Finally, the Cyclic Redundancy Check (CRC) is added at the end of the packets to ensure there are no errors in the packet.

Physical Layer On the lowest layer of the Bluetooth Low Energy core definition is the Physical Layer. It operates the radio to transmit and receive Bluetooth Low Energy data packets. Radio communication happens on 2.4 GHz unlicensed frequency. Radio uses the Gaussian Frequency Shift Keying (GFSK) modulation, where the frequency is varied to transmit a digital message. The used frequency band is split into 40 separate channels that are spaced 2 MHz apart from each others.

Direct Test Mode Bluetooth Low Energy is a unique standard as it defines the Direct Test Mode layer, which makes it possible to perform standard tests of the Physical Layer. Direct Test Mode enables the direct control of the Physical Layer allowing user to transmit and analyze different test packets.

The Host/Controller Interface The HCI provides the means of communication between the Host and the Controller. The Host can send commands and data to the Controller and The Controller can send events or data to the Host via the interface. All the traffic is capsulated in specific packet types. These packets are now explained. *Command* packets consists of Opcode, Parameter Length and the Parameters. These commands configure the controller state, request for a specific action or control a connection. For example, command may set the scan parameters. Event packets are composed of Event Code, Parameter Length and the Parameters.

Events deliver information from the Controller to the Host. Typically, an Event is a response to a Command. *Data* packets send data from the Host to the Controller or vice versa. This data typically origins from the Application Layer. These data packets comprise Handle/Flags, Data Length and the Data itself.

The Host High-level functionality of the BLE is implemented in the Host and it comprises Attribute Protocol (ATT), Generic Attribute Profile (GATT), Security Manager Protocol (SMP), Logical Link Control and Adaptation Protocol (L2CAP) and General Access Profile (GAP).

L2CAP The Logical Link Control and Adaptation Protocol is a multiplexing layer. The L2CAP channel and the L2CAP signaling commands are the two basic concepts of the layer. First of these, an L2CAP channel, is a single two directional data channel that is connected to a certain protocol or profile on the peer device. Both the Bluetooth and the Bluetooth Low Energy use the L2CAP but the BLE only implements the absolute minimum of it while classic Bluetooth utilizes most of its features. The Bluetooth Low Energy uses only fixed channels. These channels comprise one signaling channel, one Security Manager channel and one Attribute Protocol channel. All of these channels use the same frame format, which is called *the B-frame*. The frame has a length, a channel ID and a payload fields.

ATT The Attribute protocol is a Peer-to-Peer (P2P) protocol for the attribute server and the client. Attributes are stored in the server side and they are uniquely labeled. An attribute contains a value and type of the value. However, ATT itself does not define specific attributes types. The ATT also defines read and write permissions of the attribute. Design of the Attribute protocol is as simple as possible. The attribute server database is flat, which in this case means that the database can be visualized with a two dimensional table. A single attribute contains an *Attribute handle*, an *Attribute type* and an *Attribute value*. The Attribute handle (2 octets) points to the location of a certain attribute in the database. The Attribute type sets the type for the attribute, e.g., type may be a Primary Service, a Characteristic or some user defined type. Many other types of attributes also exist, such as a Secondary service or an Include type. The structure of an attribute is depicted in Figure 3.2.

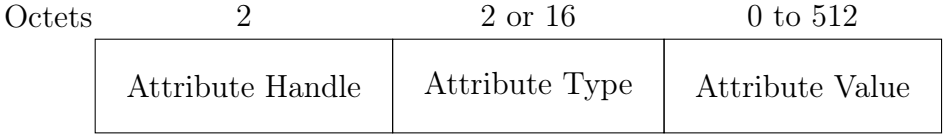

Figure 3.2: Attribute structure contains handle, type and value fields [13, p. 189]

The ATT defines six different message types [7, vol. 3, p. 520]. The first message type is a *request* from the client to the server and the second type is a *response*

message from server to the client request. These are the two main message types. Third type is a *command* sent from the client to the server. Messages with command type do not have response. Fourth type is a *notification* from the server to the client, which does not have a confirmation. Fifth type is an *indication* sent from the server to the client. Sixth one is a *confirmation*, which is a reply for indication message. It is sent from the server to the client. These six methods make it possible for both client and server to initialize communication with or without requiring response.

The Attribute protocol define several Protocol Data Units(PDUs), or protocol messages, that utilize previously mentioned types. Most of these protocol messages have both request and response type. Commonly these messages are used to discover services and service characteristics on the attribute server. It is also common to read certain characteristic value with the read request. The Table 3.2 lists all the available messages. Parentheses in the list refer to a conditional return value.

Table 3.2: Different PDUs defined by the Attribute Protocol [13, p. 219]

Message Type	**Request Parameters**	**Response Parameters**
Error	N/A	Opcode in Error, Handle in Error, Error Code
Exchange MTU	Client Rx MTU	Server Rx MTU
Find Information	Starting Handle, Ending Handle	(Handle, Type)
Find By Type Value	Starting Handle, Ending Handle, Type, Value	(Found Handle, End Group Handle)
Read By Type	Starting Handle, Ending Handle, Type	Length, (Handle, Value)
Read	Handle	Value
Read Blob	Handle, Offset	Part Value
Read Multiple	Handle	(Value)
Read By Group Type	Starting Handle, Ending Handle, Group Type	(Handle, End Group Handle, Value)
Write	Handle, Value	-
Prepare Write	Handle, Value	Handle, Value
Execute Write	Flags	-
PDU	**Command Parameters**	
Write Command	Handle, Value	
Signed Write Command	Handle, Value, Authentication Signature	
PDU	**Notification Parameters**	
Handle Value Notification	Handle, Value	
PDU	**Indication Parameters**	**Confirmation Parameters**
Handle Value Indication	Handle, Value	

The GATT utilizes these messages in predefined procedures. These procedures are explained in the GATT section next. Messages that are commonly used in these procedures are introduced here. Discovering certain type of attributes and reading their values require specific messages. Following messages are used in BLE messaging very commonly. Various other PDUs are also often used. However, they are not explained in detail in this work.

Read By Type message request reads a content of certain attributes within the given handle range [7, vol. 3 p. 500]. Read by type is useful for discovering service characteristics, or characteristics in general. If all attributes cannot be returned in the Read By Type response, the client must do a new request for remaining handles.

Read is a simple message that is used to read value of an attribute [7, vol. 3 p. 502]. Request includes the desired attribute value and the Read response returns the corresponding value of the attribute in the given handle.

Read Blob is needed to read longer attributes [7, vol. 3 p. 504]. The read blob request is used after normal Read response returns the first 22 octets, or the size defined by the Maximum Transmission Unit (MTU), of the attribute. The client will read the remaining data with the Read Blob requests by setting the offset to a position where the previous Read Blob operation ended. The Read Blob request is repeated until there is no more data available. In case the attribute length is not fixed and the last packet is full, the client cannot know that there is no more data available and makes a new blob request. In such case server returns an error message to inform the client. Blob in Read Blob stands for a *binary long object*.

Read By Group Type is similar to Read By Type message type but it can only be used to discover grouping attributes [7, vol. 3 p. 507], such as Primary Services. Grouped attribute contains number of different attributes which are located in consecutive handles in the attribute server. Grouped attribute is then defined by the *handle range* in the attribute database. The Read By Group Type request takes the attribute type and searched attribute range as parameters. If the range is set from 0x0 to 0xffff, the whole attribute server database is searched for a selected grouping attribute. The type defines the desired attribute type. For example, the value must be set to 0x2800 (Primary Service identifier), if the client wants to search for Primary Services. The response message returns the desired attributes and associated handle ranges on the defined range.

GATT The Generic Attribute Profile is based on the ATT, which introduced the client-server separation. The purpose of it is to connect two devices together. The GATT dictates how two devices exchange messages. The message exchange is carried out by certain procedures that are divided into three categories: discovery procedures, client-initiated procedures and server-initiated procedures [13, p. 231]. With discovery procedures, client identifies the primary services. Discovering primary services provides the attribute database grouping. After grouping, the client can move on to identify the structure of individual primary services. The range of a service is defined by a handle range. A Primary service may contain secondary services, characteristics and descriptors. After knowing what information the device holds, the client can continue and perform *Client-Initiated Procedures*. With these

procedures, the client can read or write device characteristics or their descriptors. The opposite actions of client initiated procedures are *Server-Initiated Procedures*. There are two types of server initiated procedures, first of them is notifications. A notification is a server originated message that is sent to the client at any time. An indication is the second type of server initiated messages and it is very similar to a notification. Unlike notifications, indications use flow control and they must be confirmed by the client before sending a new indication. Usually client configures the server to send server-initiated messages on certain events to reduce the network load.

There is one exceptional procedure that does not belong to discovery, client-initiated nor server-initiated procedures. It is the *Exchange MTU procedure*, which defines the MTU size for subsequent messages [13, p. 232].

SMP The Security Manager Protocol is responsible of generating encryption keys and identity keys for BLE devices. SMP is a peer-to-peer protocol and it uses dedicated L2CAP fixed route (see Figure B.1 in Appendices) for the data exchange. The SMP block also stores the encryption and identity keys of other devices. Block utilizes the interface of the Controller to directly provide stored encryption and authentication keys for encryption and pairing processes.

GAP The General Access Profile defines general routines of Bluetooth Low Energy. It defines how to discover devices, how to connect to a device and how the information is presented. Moreover, the GAP block defines *bonding* procedure, which is a method for permanent relationship. The block also introduces a method of using private addresses which is useful to hide the device from unwanted connections.

The Application Layer On top of the Controller is the Application Layer. The Application Layer is the highest layer in the Bluetooth definition. The layer defines three specifications: characteristics, services and profiles [13, p. 36]. These specifications are built on top of the GATT.

A *Characteristic* is a piece of data which is in known format. Characteristic is labeled with a Universally Unique Identifier (UUID). Characteristics are reusable, which limits it to not to have any behavior. The characteristic is presented in computer readable format on the device.

A *Service* is a set of characteristics in human readable form. Service includes the server side behavior associated to the characteristic. Services can include other services but the parent service is incapable of changing the attributes inside the included service. There are two types of services: primary and secondary. Primary services express the functionality of the device. Secondary services assist primary services. Services merely define the behavior when characteristic is operated and do not describe, e.g., how devices connect or what GATT procedures are being used to find the service or its characteristics. In other words, services do not describe the functionality but they rather specify features of the device.

A *Profile* is the highest abstraction concept in the Bluetooth Low Energy stack. Profiles are specifications and they outline the device functionality. This specifica-

tion includes one or more services on a device and it must describe how at least two devices communicate. The profile specifies how a certain device is discovered and connected. In addition, profile defines how client finds services and service characteristics and how to use the service to enable the device functionality for a certain use case.

3.2.2 IEEE 802.15.4 standard for Wireless Personal Area Networks

The Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard is designed for low range, low power consumption radio communication applications, i.e., Low-Rate Wireless Personal Area Networks(LR-WPANs) [8]. Standard defines the Physical layer and the Datalink layer of Open Systems Interconnection (OSI) model. Higher layers must be implemented individually to obtain a higher level of abstraction in communication. For example, famous ZigBee standard is implemented on top of the 802.15.4 standard. In addition to the ZigBee, the IP protocol can be set up on top of the standard.

A device can use 64-bit IEEE address or 16-bit short address as an identifier in 802.15.4 network. The short address should be allocated by the Personal Area Network (PAN) coordinator when the device is associated to the network. An extended address may be used for direct communication. 802.15.4 defines two different device types: a Full Function Device (FFD) and a Reduced Function Device (RFD). A FFD implements the standard completely and it can be used as a PAN coordinator. In the coordinator mode, the device is capable of setting up an 802.15.4 network to which other RFDs can join. There must always be a coordinator in the PAN. Reduced function devices are only capable to communicate through a common coordinator node.

Applicable network topologies The IEEE 802.15.4 standard offers two different network topologies: Star and Peer-to-Peer (P2P) topology (Figure 3.3) [8, p. 8]. In the star topology, every node is communicating through a central device, the PAN coordinator. Therefore nodes cannot communicate directly with each other and in many cases, there is no need for nodes to exchange information with leaf nodes. If there is a need for direct communication, P2P topology may be a better option for the network topology. In P2P topology, any device is able to communicate with other device in range. Peer-to-peer communication enables the implementation of more complex network formations, such as mesh topology. P2P topology network allows message delivery via multiple hops. However, more FFDs are needed since RFDs cannot communicate directly with each other.

The standard defines two lowest layers of the OSI model. The Physical layer is the lowest and the Datalink layer the second lowest layer in the OSI model. Layers are accessed through different Service Access Points(SAPs). Both of the layers have two SAPs with different purposes. Upper abstraction layers are implemented on top of the MAC sublayer. In a typical implementation of the 802.15.4 standard, only the MAC sublayer services are visible to the upper layer. Later part of the text will discuss the functionality of these layers and define their purposes and features.

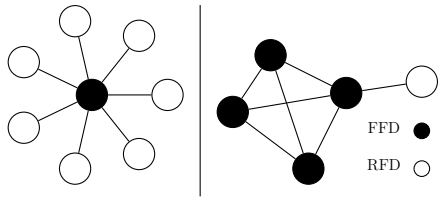

Figure 3.3: Network topologies: star topology depicted on the left side and P2P topology on the right side. [8, p. 9]

Data transfer transactions There are three types of different data transfer transactions in 802.15.4 [8, p. 13]. In the first one the data is transferred from arbitrary device to a coordinator. The second transaction is the opposite of the first transaction type. The data is transferred from a coordinator to a device. Third type of transfer is between two peer nodes. Such transaction only takes place in P2P network. The exact mechanism of each transfer type is further specified by whether the network utilizes periodic beacons or not. A *beacon-enabled* PAN is used in networks that must be synchronized, e.g., in a network where every node must send data to the coordinator during a certain time window. In general, synchronization reduces latency since there will not be as many collisions. If beacon is not required, the network can work in a *non-beacon* enabled mode. However, beacons are always used in the network discovery regardless of the type of the network.

In the non-beacon mode the data transfer can occur at any time. In *device to coordinator* transaction, a device may simply send a data frame to the coordinator, which can optionally respond with an acknowledgement frame to complete the transaction. In *coordinator to device* transaction type, the coordinator awaits the device to make contact and request the data. The device requests data with a specific request data frame. If the frame is pending, the coordinator transmits the data frame after acknowledging the request frame.

In the beacon mode, the data transfer from devices to coordinator takes place inside superframes. Coordinator may transfer data to devices by making an indication of pending data in the network beacon frame, if there is need to initiate a transfer from the coordinator to a device. Then, the device will notice there is pending data available and it will make a data request to the coordinator which begins the data transmission.

Beacon superframe The 802.15.4 standard defines a superframe structure (Figure 3.4), which can be used in the PANs for data transmission. The format of the superframe is decided by the network coordinator. The frame is bounded by beacons, which define the duration of the frame and they are sent by the coordinator [8, p. 12]. The superframe inside the beacon is divided into 16 equally long slots. This time is called an active period. After the active period, the frame can optionally have an inactive period. The coordinator can enter a low-power mode during the inactive period. Inactive period also allows other coordinators belonging to the network to create their own nested beacons inside the superframe beacon. This is an extremely

useful feature for cluster tree networks. However, in large beacon enabled networks, the latency is higher [8, p. 10] but on the other hand it is also predictable unlike in similar network that does not utilize beacons. Use of superframes is optional. They are used to synchronize associated devices and also while identifying the PAN.

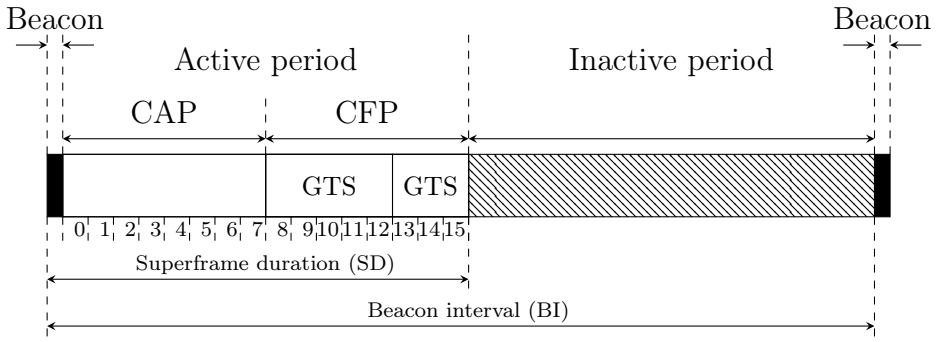

Figure 3.4: General structure of a superframe with two GTSs inside CFP. Derived [8, p. 20]

Devices wishing to transmit data are competing for access to the medium during the Contention Access Period (CAP). For the channel access, nodes may use either slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) or ALOHA mechanism [8, p. 15]. ALOHA is used only in very simple networks in which medium is thought to be accessible at any time. CSMA-CA method utilizes the CCA service of the PHY layer to perform a check whether the channel is available. The CSMA-CA method is described later in more detail. The CFP starts immediately after the CAP on the slot boundary and it ends at the end of the active portion of the frame. All GTSs are located inside the CFP. Guaranteed time slots may reserve contiguous slots from CFP. The CSMA-CA is not used with GTS since the slot should be dedicated to a particular node. To request a GTS, node can utilize the MAC Layer Management Entity (MLME) service to send a GTS request to the PAN coordinator.

CSMA-CA The CSMA-CA algorithm enables the device to access the channel medium without causing collisions in the air [8, p. 14]. It is performed in the transmission of data or with MAC command frames that are transmitted within the CAP. Because of the nature of these two transform methods, there are also two different CSMA-CA variations. In both cases these two variations of the algorithm are based on checking whether the channel is available. If the channel is reserved, the device waits for a moment. The wait time is referred as the backoff time. In slotted version of the CSMA-CA algorithm, the backoff period boundaries are aligned with boundaries of superframe slots. In the unslotted version, backoff periods are not related to other devices in the network.

The algorithm uses three variables that are stored on the device. *NB* is the number of times the algorithm was required to backoff while attempting to transmit. *CW* is the contention window length and it defines the number of required backoff

periods to accomplish the transmission. CW is only used with the slotted version of the CSMA-CA. *BE* is the backoff exponent, which denotes the amount of backoff periods that the device must wait before trying transmission again.

PHY layer The physical layer, or the PHY, is a layer between the MAC sublayer and the physical radio channel. It is an interface, which the MAC sublayer utilizes to transmit messages with the physical radio. The physical layer provides the PHY data service and the PHY management service. The data service transmits and receives PHY protocol data units(PPDUs) across the physical radio channel [8, p. 31]. The Layer management entity is called the Physical Layer Management Entity (PLME), which provides the layer management services and a SAP to access it. The PLME also maintains a database of objects that belong to the physical layer. This database contains attributes, including the used RF channel, transmit power and used Clear Channel Assesment (CCA) mode. The database is called the PHY Personal area network information base (PIB). Data services and management services of the PHY layer are accessed via two different SAPs. The definition of the physical layer consists of [14]:

- Mode selection of the receiver. The radio transmitter has three different states which are transmission, receive and sleep state. Radio is switched off while in the sleep state.
- Energy Detection, ED is an estimate of the present channel signal power. It is intended to be used on the network layer as a part of the channel selection. There is no need to decode the signal since only the signal strength is measured.
- Link Quality Indication, LQI measures the quality of a received packet. LQI can utilize Energy Detection, signal-to-noise ratio estimation and combine these methods. LQI measurement is performed for every received packet. The value of LQI varies between 0 and 255.
- Clear Channel Assessment, CCA is a routine to check whether the selected channel is currently available for transmission purposes. In CCA Mode 1, channel is available if Energy Detection value on present channel is currently below defined threshold. CCA Mode 2 reports busy medium only if the signal is compliant with 802.15.4 standard with the same modulation and spreading characteristics that the device is currently using. The signal strength may be above or below the ED threshold. CCA Mode 3 combines the mode 1 and mode 2 with logical AND or OR operator. Fourth mode reports idle medium at any time and it may be used in simple networks with low traffic. There are two more CCA modes that report busy medium upon detecting preamble symbol but they are not covered in this context.
- Channel selection from 27 channels of 802.15.4 standard. Channels are separated into 3 different frequency groups. Channels from 1 to 10 work on range of 902 to 928 MHz and channels from 11 - 26 operate on 2.4 – 2.4835 GHz range. Channel 0 uses 868 MHz and it is used in Europe.

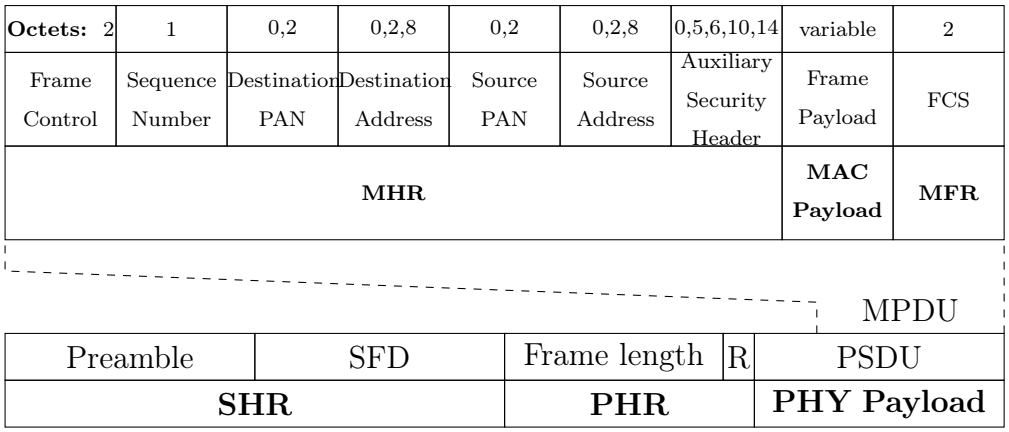

Figure 3.5: PPDU and MPDU packet formats. Derived [8, p. 160][8, p. 57]

The PPDU packet structure is presented in the Figure 3.5. The PPDU is composed of a Synchronization Header (SHR), a PHY Header (PHR) and the actual payload [8]. The packet is received by the physical channel from left to right so that the preamble is received first. The length of the preamble is 4 octets, and with O-QPSK modulation all the bits are zero [8, p. 160]. The SFD field indicates the end of the SHR field. Length of the SFD is one octet and it has a predefined value. Seven bits are reserved for frame length field and it indicates the size of the payload in PHY service data unit (PSDU). One bit is reserved after the frame length field. The length of the PSDU varies between 0 and constant *aMaxPHYPacketSize* (= 127) defined by the standard. The payload carries the MPDU. First four octets of the PSDU are reserved and next octet is for MPDU acknowledgement. Octets from 6 to 8 are again reserved. Octets from 9 to *aMaxPHYPacketSize* are actually used for MPDU. Note that PSDU field does not equal to MPDU. It only contains the MPDU [8, p. 161].

MAC sublayer The Medium Access Control (MAC) sublayer is a part of the Data Link Layer (DLL) of OSI model. The MAC sublayer is located on top of the physical layer and it is a functional interface between user-defined application layer and the PHY layer. The MAC Common Part Layer (MCPS) provides the actual functionality of the sublayer, which is listed later. The MLME provides services through which the layer may be managed. These services include association, disassociation, scan services and reset of the layer. Like the PHY, MLME of the MAC sublayer contains database, which is referred to as the MAC sublayer PIB. Higher layer can access the MLME through MLME-SAP. The MAC data service is accessed through other SAP, which is known as the MCPS-SAP. Moreover, there is an interface inside the sublayer that enables the MLME to use MAC data service. The MAC sublayer provides: channel access mechanism, beacon management, GTS management, frame validation, acknowledged frame delivery and PAN association and disassociation [14].

The General Mac frame format (MPDU) is visible in Figure 3.5, opened from

PSDU [8]. Messages are transmitted in MAC frame format inside the MAC sublayer. The MAC frame comprises the MAC Header (MHR), the MAC Payload and the MAC Footer (MFR). The header unit consists of a frame control, a sequence number, addressing fields and an auxiliary security header. *Frame Control* field contains information of the frame type, addressing fields and various control flags, such as security. Frame type may be one of the following: beacon, data, acknowledgement or MAC command. Information about addressing fields decides in which format the address is given, if in any (destination fields may also be non-existent), in destination address, i.e., short (16 bit) or extended address (64 bit). The *Sequence Number* field is needed for specifying the sequence identifier of the frame. The sequence is used to match corresponding frames, e.g., it is used to match request and acknowledgement frames or beacon frames. The *Frame Payload* field of MPDU contains the actual message of the packet and it is specific to every individual frame. The payload is encrypted if the security flag is enabled in the Frame Control field. Last element of the MAC frame is the Frame Check Sequence (FCS) that is calculated over the MHR and payload part of the frame. Message integrity is ensured with FCS.

3.2.3 HTTP and CoAP protocols

The Hypertext Transfer Protocol The HTTP is a stateless application-level data communication protocol [15] ordinarily built on top of the Transmission Control Protocol (TCP). Originally it was designed to be an interface in distributed object systems [16]. Data exchange is based on requests and responses; a client makes a requests to a server to which the server responds with a response message. An HTTP client is a program which establishes the connection to the server [15, p. 7]. The server, in turn, is a program that accepts client connections and responses to client requests [15, p. 7]. For stateless protocols, each request and response is treated independently and requests, or responses, are not related with each others. Stateless protocol itself does not store any data that request or response packets contain. Currently, version 1.1 is in the use but HTTP/2 proposed standard was released in February 2015. The new version of the HTTP will improve performance of the protocol [17].

Addressing of HTTP requests is based on the Uniform Resource Identifier (URI). Target of an HTTP request is known as a resource. The purpose of the URI is to define the exact location of the desired resource on the server. A resource contains the required data that is fetched with a request. How the resource is presented for the End User is called a representation of the resource. Creation, deletion or manipulation of resources are also possible with HTTP requests.

Intermediaries in HTTP Intermediaries are placed in between of the User Agent and the resource origin. Commonly intermediaries are used as proxies, gateways or tunnels. Combining of these types is possible and single intermediary may have multiple roles. Intermediaries can also be chained together in which case the request is delivered to the origin server via many middle hands.

Proxy is a message-forwarding agent that receives requests containing the absolute URI [15, p. 10] and passes them forward. Proxies are ordinarily needed to group traffic of certain organization to trough a mutual intermediary for security reasons.

Gateway acts as an origin server for connections towards outbound connections. A gateway can communicate with inbound servers through any arbitrary protocol, e.g., Constrained Application Protocol (CoAP) or BLE. Gateways can encapsulate services which would otherwise be difficult to utilize from the point of the End user. Data caching is an important feature of the gateway and it is an effective method to remove unnecessary load of inbound servers.

Last defined intermediary type is *tunnel*. A tunnel is an invisible intermediary that relays the message without changing it.

Caches A local storage of data that is formed from received messages is called a cache. Caching subsystem governs the storing, retrieval and deletion of the storage. Caching improves response time of the system and reduces the load of the network. It is convenient to use caches when resources cannot be obtained easily. For example, if a network consists of a gateway, which encapsulates inbound servers, and an external User agent that makes requests inside the network. The user agent may request data from inbound servers via the gateway. The gateway can have already unchanged cached data of inbound servers and it is able to send response message immediately with very low response time instead of using time to forward the message. However, the cached representation may be obsolete. For the same reason caching can be hazardous in some cases.

HTTP methods Method type is the most essential token of the request. It reflects the purpose of the request and describes which kind of action is to be taken on the origin server. Method type is defined in the beginning of the request with the protocol version definition in order to early determine the semantics of the message. All of the methods are conventionally defined in uppercase letters. Common request methods with brief descriptions are given in Table 3.3. List does not contain every available method, including partial update method PATCH.

Messaging A typical request contains the request method type with a resource identifier, a used protocol version, information about the user agent and the name of the host. The request method type dictates the type of the action, which server performs to the resource that is associated with the URI. Used protocol version field defines the version of the HTTP. The user agent refers to the application, which is utilizing the protocol, e.g., general-purpose browser or command-line program [15, p. 8]. The Host field in the HTTP request contains location information about the host. In addition to preceding headers, the request may contain various other fields.

Response sent by the origin server contains three-digit status code which indicates the nature of the message. Status messages are divided into five categories. First digit of the code denotes the category and remaining digits refine the meaning. Status codes 1xx are informational and mean that the request was received and the

Table 3.3: Common request method types [16, p. 22]

Method	Server action
GET	Request the current representation of the targeted resource.
HEAD	Similar to GET, but only transfer the status line and header section of the response.
POST	Perform resource-specific processing on the request payload. May create a new resource.
PUT	Replace all current representation of the target resource with the request payload. Can be also used to create a new resource.
DELETE	Remove all current representations of the target resource.
CONNECT	Establish a tunnel to the server identified by the target resource.
OPTIONS	Describe the communication options of the target resource.
TRACE	Perform a message loop-back test along the path to the target resource.

process is continued. Class 2xx status codes mark that the request was received, understood and accepted. Codes 3xx mark redirection and further action is required to complete the request. Lastly, 4xx status codes indicate a client error and 5xx codes a server error.

The Constrained Application Protocol The CoAP is a web transfer protocol for constrained nodes often equipped with a relatively low speed microcontroller and a limited amount of memory [18, p. 1]. The protocol is especially intended for Machine-To-Machine (M2M) applications, including building automation. Numerous implementations of CoAP servers and clients are available, such as Erbium for Contiki Operating System. Like HTTP, CoAP provides the request–response interaction scheme for data exchange. Client sends a request with a method code to perform an action on a resource on the server. Server performs the action and responds to the request with a response code and possible representation of the resource. However, unlike HTTP, CoAP utilizes asynchronous data transportation over a datagram-oriented transport layer, such as User Datagram Protocol (UDP).

CoAP can be illustrated with two-layered approach. *Messaging layer* deals with the asynchronous nature of CoAP and complements data-oriented transport layer. In messaging, CoAP definition includes four types of messages: Confirmable, Non-Confirmable, Acknowledgement and Reset. *Requests and Responses layer* is the higher layer and describes the request–response scheme.

Messaging model Exchange of messages between endpoints is described in the CoAP messaging model. In the messaging model, the CoAP uses 4-octet binary headers (depicted in Figure 3.6) which may optionally be followed by options and a

payload. Each message contain identifier for duplicate packet detection. Identifier is also used to improve robustness of the network since CoAP does not utilize transport protocol, such as the TCP, to ensure the transportation. Reliability is improved with setting the message type to Confirmable. A Confirmable message expects confirmation from the target endpoint and it is retransmitted if Acknowledgement message is not received in certain time. Server may also respond with an empty Acknowledgement message if it cannot reply to the client immediately. Client retransmission is halted and the server can send the actual response later with different Message Identifier to which client responses with an Acknowledgement. Messages which do not need reliable communication and do not require retransmission may be sent as Non-Confirmable messages. These messages also include a message identifier for detecting duplicate packets, but they are not acknowledged by sending a separate confirmation message. A reset message may be sent if the recipient is not able to process the Non-Confirmable message.

Request and Response model Request and Response model is placed on top of the messaging model to carry the semantics of the message and the message itself. Meta information includes, among other things, either a Method Code or Response Code, Message Identifier and the default or optional information of the message. CoAP request methods are similar to HTTP except only the basic types are supported: GET, POST, PUT and DELETE. Next part of the text inspects the structure of the message and provides detailed view of CoAP requests and responses.

Figure 3.6: CoAP Message header [18]

Message structure A CoAP message comprises a header, a token, options and a payload. The CoAP message header is manifested in Figure 3.6. The header length is fixed and it is binary encoded for efficiency. In total, there are five different fields in the 4-octet message header. The first octet contains the information about protocol version and the message type. It also defines the length of the token. First two bits of the header are reserved for CoAP *version number* (Ver). Next two bits after the version define the *type* (T) of the message with following values: Confirmable (0), Non-confirmable (1), Acknowledgement (2) and Reset (3). Next in the header comes the *Token Length* (TKL) that is half octet integer and it assigns the size of the token. Token length can vary from 0 to 8, which it must not exceed even it is possible due to the size of this field. TKL defines the length of the token in bytes. Sizes greater than 8 are reserved for other purposes. Second byte is fully reserved for the *Code* field, which is split into 3-bit class and 5-bit detail sub fields. Message code categories are comparable with HTTP response statuses (described earlier) despite the fact that also request messages contain code in CoAP. Values of class codes are

following: Request message (0), Success (1), Client Error (4) and Server Error (5). Other class values are reserved. Message is displayed c.dd format, where c is the message class and dd the detailed description of the class. For instance, message code 0.00 indicates an empty message. Request messages always use code field with value 0.xx where "xx" defines the type of the request method. Currently, only values from .01 to .04 for GET, POST, PUT and DELETE are defined. Last part of the header is two octet *Message ID*. It is an identifier which separates different messages from each others to prevent duplicates and match different messages. Token value, which length is already defined by TKL, follows the header. Other options come after the header and token. Last element of the message is the optional payload related to the targeted resource.

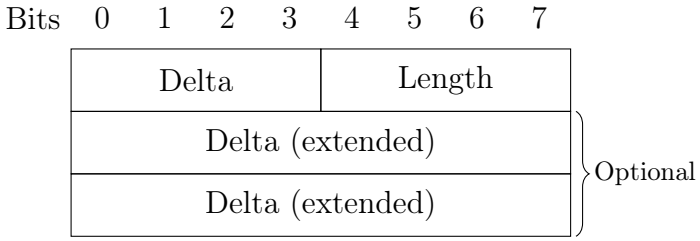

Figure 3.7: CoAP message option structure. Options come right after the Token, or the Header if TKL is zero. [18, p. 18]

Messages can contain a list of options. Options may be used with both requests and responses. Option, presented in Figure 3.7, is also binary coded. The type of the option is obtained relatively with the *Delta*. When receiving a message, the message type counter is zero. Delta increases this counter. Because of the delta feature, options must be declared in order. For example, *Uri-Path* option number is 11 and *Block2* number is 23. To express these options in the message, Uri-Path option must be declared first in the options list with a delta value 11. Block2 is then declared with delta value 12 (11 + 12 = 23). In the request, options contain the resource identifier which addresses to the desired resource.

There are two types of options: critical and elective. If the option type is *elective* and the end point do not recognize the option, it must be silently ignored. However, if option is set to *critical* and it is unrecognized, the end point must return 4.02 (Bad Option) in the response with a diagnostic of the unrecognized (or possibly malformed) option. Option also contain *UnSafe* flag. UnSafe flag determines how proxy should handle an option if it does not recognize it. These flags are not directly available in the option, but are generated from the message type number with a bit mask [18, p. 38].

Messaging Confirmable and Non-confirmable message types of the Code field in the header initiate a request. If needed, the server responses to the client request after receiving and handling processes. The response is matched by the request token, which is generated by the client. Message ID matches Confirmable message

to its corresponding Acknowledgement response message. In the basic case, server receives a Confirmable message from the client and responds to it directly with Acknowledgement message containing the requested information. Such transfer scheme in CoAP is called a *Piggybacked Response* (Figure 3.8a). In all cases it is not possible to immediately respond with a Piggybacked response message in which case the server sends an Acknowledgement response with an Empty Message code (Figure 3.8b). Server can then take the needed time to obtain the resource without client repeatedly retransmitting requests. After the resource has been obtained, the server sends the response to the client not as a new Acknowledgement message but as a separate response message matched by the token, which remain unchanged.

Message types are abbreviated as follows: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) and Reset (RES).

Request may also be carried as a Non-confirmable message. Server can respond to such message with a new Non-confirmable message. It is also possible to server respond with a Confirmable message which requires the client to confirm the response message.

Each message may carry a token. The length of the token is defined by the TKL (Token Length) in bytes. The Token matches the response message with a request. Message ID itself is not adequate for message matching purpose, though it matches Piggybacked messages with request with token. However, other types of messages need Token based matching (Figure 3.8).

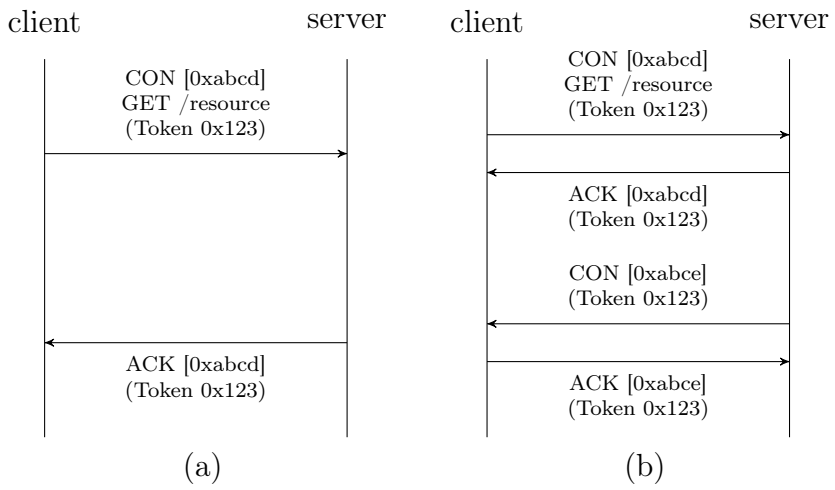

Figure 3.8: Server can immediately respond to the request from the client with a Piggybacked message (a) or later with separate response (b) if time is required to obtain the resource

Subscribing to a resource A major problem with Representational state transfer (REST) based CoAP server is that if client wants to keep track of periodically changing resource, it must constantly request a new version of it. Polling consumes energy and is not a good way to implement such feature. Therefore CoAP

introduces resource observation with still maintaining the REST features [19].

When client wants to subscribe to a certain resource, it transmits an ordinary Confirmable message to the server with an *Observe* option. This option tells the server that client wants to listen to the changes of the resource and receive a notification whenever these changes happen. Server includes the client to a listener list and sends a notification whenever changes occur. Resource observation removes the unnecessary load of the network and it also makes the design simpler.

3.3 Design approaches and implementation techniques

3.3.1 REST and Resource-oriented architecture

Simply put, a resource is merely a piece of information that usually forms a logical entity on the server end. Resource-oriented Architecture (ROA) is a term introduced in recent years that describes certain type of software architecture where the stored data is thought as resources [20, p. 80]. Sometimes REST and ROA are used as synonyms. However, REST is not a software architecture but more of a set of design criteria [21, p. 76]. In addition, REST is not necessarily bonded with HTTP protocol, even it is often associated with it, nor it employees the URI structure, and according to the definition by Richardson et al. [20], ROA limits the scope to general web services that utilize HTTP and URI. It is good to keep in mind that the concept of ROA is fresh and binding it strictly only to the HTTP is a matter of opinion.

Representational State Transfer In software design, an architecture describes the structure of an application. Representational state transfer (REST) is a set of architectural guidelines that delineate the structure of the application. More precisely, REST lists a set of design criteria [21, p. 76], which the architecture must satisfy.

REST Constraints The first of these constraints separates the application to client and server. Client-server architecture is natural for web services and it is easily satisfied. Next constraint concerns the stateless nature of communication. Each client request to the server must contain all necessary information to complete the request without taking advantage of previously stored context. Stateless data transfer improves reliability, visibility and scalability. There is a trade-off with network performance with stateless transmission since the same data is sent repetitively. To improve the efficiency of the network, the REST defines client caching constraint. Response message data may be cached on permission. Cached data can be reused later to avoid unnecessary requests and load of the network. Client side data caching may cause data to become obsolete.

Perhaps the most important requirement, or feature, of REST is the concept of *Uniform Interfaces*. Uniform interfaces generalize system components and simplifies the architecture. Without uniform interfaces, it is complicated to understand interfaces and every service must be designed and implemented completely individually.

Uniform interfaces are the base for transparent message interchange. Generalization is emphasized in uniform interfaces.

Constrained RESTful environment In constrained network environments, network nodes typically have limited amount of processing power. Usually these nodes are equipped with low power microcontrollers and limited amount of memory. The Constrained RESTful Environment (CoRE) definition particularizes the REST requirements in these constrained networks [22]. The discovery of resources is important aspect of the definition. The discovery procedure provides URIs a URI */.well-known/core*, which is the default entry point for requesting the list of resources on the server [22, p. 3].

ROA based approach in Web service design As referenced previously, ROA scopes the implementation of REST to exploit Web technologies, i.e., HTTP and URI scheme [20]. An URI is composed of the address and the name of the resource. For the sake of addressability, every resource must be pointed by at least one URI or otherwise it is not reachable. It is good manner to make URIs as descriptive as possible. For example, in this work, gateway contains scanned sensors from the environment and corresponding cached virtual from the cloud. Each physical and cached virtual sensor represent a single resource that can be uniquely identified by the Sensor Unique Identifier (SUID). These resources are organized so that cached cloud sensor resources are available at */sensors/cloud/{suid}* and locally scanned physical sensors are located at */sensors/ble/{suid}*.

Addressability is one of the main features of ROA. Addressable data can be accessed immediately without performing any preliminary actions. Addressable application reveals all the essential data to the web as resources, which can be pointed with a URI. The end user can access any piece of information on the server and manipulate it. In the sensor example given before, every sensor is addressable. Statelessness is the second major concept in ROA and it is inherited from the REST.

REST constraints define how the Uniform Interface works and what functionality it has leaving open the question of how the interface should be implemented in HTTP. The HTTP standard describes different request methods but it is still inadequate itself to explicitly dictate how the resource manipulation should be carried out in HTTP and which methods should be attached to different manipulation operations. ROA defines the Uniform Interface more elaborately. Next part attaches HTTP methods to different resource operations.

- **GET** method receives the representation of the targeted resource from the server. It is perhaps the most used method in HTTP. For instance, the end user can receive the current representation of the sensor with id 12345 on the server by making GET request to a resource that is addressed with following URI: */sensors/ble/12345*. The server will respond with the latest representation of the wireless sensor which, for example, includes the rssi value and the battery level. On some applications, GET can be used to request representations from

the past. According to addressability, all the information should be reachable with GET requests.

- **PUT** request creates new resource that is based on the information given in the request. If the resource already exists in the addressed URI, it is overwritten (if allowed) with new data. Resource is created to where the request URI points unlike in the case of POST method.
- **DELETE**, like mentioned in HTTP standard, removes the resource on the server. DELETE request to */sensors/ble/12345* will result in the deletion of the sensor resource.
- **HEAD** requests are similar to GET but they only fetch the metadata of the resource. HEAD is much faster than GET since the entity body is not transferred. It can be used to test whether the resource exists and to find information about the resource. HEAD can be very useful when working with low power constrained nodes with CoAP.
- **OPTIONS** tells the client which operations are available for the requested resource. OPTIONS response header contains the Allow field which lists all the allowed methods. OPTIONS provides useful information of the resource which can be used for client configuration. OPTIONS has not yet been standardized well and many REST applications do not support OPTIONS method. Let's say resource */sensors/cloud/12345* can be created, deleted and modified. Requesting OPTIONS for the resource in question would return response, which contains header field Allow with value GET, PUT, DELETE, OPTIONS (and other possible methods).
- **POST** substantially serves two purposes of which one is out of the scope of REST and concerns remote procedure calls. POST resembles PUT by its function but there are noticeable differences. The main difference compared to PUT is that when creating a new resource, post does not allow the request to decide the resource URI.

Table 3.4: Comparison by example of PUT and POST requests. Suggested usages of requests

	PUT, new resource	**PUT, resource exists**	**POST**
`/sensors`	N/A	N/A	Create a new sensor category
`/sensors/cloud`	Create a new sensor category	Modify category parameters	Create a new sensor
`/sensors/cloud/1234`	Create new sensor	Modify the sensor	N/A or append metadata

Resource safety concerns Method safety is an important aspect of ROA. Methods that are considered safe do not change the server properties, i.e, create new resources or modify them. Request only asks for information about resource. GET, HEAD, OPTIONS are generally considered to be safe. Requesting representation of */sensors/ble/12345* with GET does not change the resource in any way. Idempotence is the second important aspect in the resource safety. Request is idempotent if it only changes the resource once even the same request is made multiple times. Multiplying value by one is both safe and idempotent, since no matter how many times the operation is repeated, there will not be any changes to the value. However, multiplying by zero is only idempotent. Value changes when operation is applied first time, but multiplying zero by zero does not change the result and therefore zero multiplication is idempotent.

Idempotent requests are safer than non-idempotent ones. End user can mistakenly make duplicate requests to a resource and if the request is not idempotent, the resource will be unwillingly modified. Idempotent request changes resource attributes by defining new absolutes values. For example, relative request would tell the resource to increment certain value. Idempotent request first requires the current value of that certain value of resource and after obtaining the information, request can tell the resource to set change the value to received absolute value, which was incremented in client side. Even if the request was performed many times, it would not result in any changes after the first request. However, relatively changing resource attributes would result in changes everytime new request was made. That is the reason why requests should be idempotent and if possible, resources should not change on the origin server. In general, PUT and DELETE are idempotent methods. PUT request, with an assumption that resource values are changed in absolute manner, can be applied many times to the target URI and it will only have effect on the first time. Same applies for DELETE: the resource is deleted on the first request and requests after that do not change the state of deleted resource, it still remains deleted.

Safety and idempotence make the application reliable, even if the network is not reliable. If the request was received by the server but the client never received response, it is safe to make a new duplicate request. On the aspect of ROA, POST method is not either safe or idempotent. In a well designed uniform interface, all services work in a similar way and use same HTTP methods for same purposes.

3.3.2 Node.js

Node.js is a powerful tool for building network applications [23]. Node.js runs on V8 Javascript Engine made by Google. Inherent to Javascript, Node uses non-blocking IO and asynchronous events allowing the process to run normally at all times when blocking IO process has to wait each function to complete [24, p. 36]. In Node.js environment, it is common to execute a callback function upon completing certain function. The callback function is passed as a parameter to the function that is to be executed. The callback function is called internally with appropriate parameters to handle the function completion in the end of function execution.

Node.js community offers vast selection of modules. Modules can be thought of as software components, they contain a set of functionality which is accessed through its interface. Node.js environment is designed for developing Data-intensive real-time (DIRT) applications [25]. Node itself is very lightweight platform with a small memory footprint [25].

Node-RED Node-RED environment is built on Node.js and it enables flow based programming in the Node.js environment [26]. Node-RED installation provides standard set of reconfigurable, reusable nodes that are connected to each other graphically in a browser-based user interface. Additional nodes can be created with javascript to supplement the functionality.

Node.js environment in embedded systems There exist various different development platforms that are capable of running a Node.js environment. Beaglebone Black development board [27] with Debian distribution is used in this work.

4 The condition monitoring system

The main goal of this work is to automate condition monitoring of wear plates in quarrying processses. This section describes the design and the implementation of a system that aims to automate the condition monitoring. Firstly, a general overview is given to describe the purpose and features of the monitoring system. In the overview, system is divided into three layers: sensor level, gateway and cloud. These different system parts are later discussed in detail.

4.1 Description

Monitoring system is divided in separate layers to clearly distinguish separate systems from each others. Some of the components in the design are already implemented which must be accounted in the design as a set of restrictions. These already implemented system components include the cloud service with HTTP interface and 802.15.4 standard sensor. The interface of the cloud service forces the HTTP server on gateway to function in a certain way. Moreover, the 802.15.4 sensor and its design environment strongly guides to use the CoAP protocol for data transfer. Design overview also includes mobile application, but it is only covered briefly in this work.

4.1.1 Overview of the system

This section describes the design of the wear plate condition monitoring system. Figure 4.1 divides the system in three layers. Lowest element of the system is the wireless sensor network consisting all of the Bluetooth Low Energy and 802.15.4 based sensors. The sensors are independent and they do not need to communicate with each other. These sensors duty is to measure the acceleration of the plates and send the gathered data to the gateway.

Gateway contains modules which are capable of communicating with sensor network protocols. Bluetooth Low Energy module of the gateway functions as the central node, which forms all the connections with BLE sensors. The 802.15.4 module is the PAN coordinator of the 802.15.4 network. The network does not utilize beacons. However, the gateway node identifies the network and distributes IPv6 based addresses to the nodes. According to the 802.15.4 standard, network identifying always exploits the network beacons. To interact with cloud service, gateway utilizes Application Programming Interface (API) of the cloud service provider. This API is capable of doing HTTP requests to the cloud. Moreover, the gateway has an HTTP server to provide user interface for monitoring and debugging.

Cloud is the highest level of the system. Cloud has the database for storing the sensor data. Cloud also provides the service to analyze data. Cloud does not provide service to analyze the data. However, it provides Node-RED environment, which is used to implement the data analysis. Cloud is accessed via its HTTP interface with Javascript API.

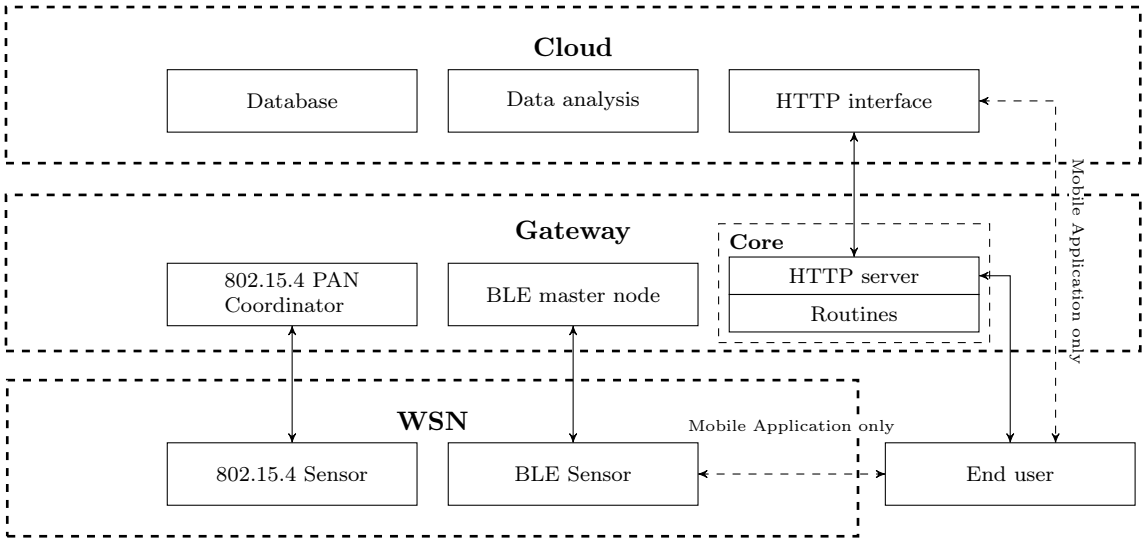

Figure 4.1: Overview of the system. Internal connections of the blocks are not shown

4.1.2 Sensor

Two different sensor boards were tested in the implementation phase. First sensor contains 802.15.4 standard radio and second uses BLE for communication.

Common functionality Both sensors use LIS3DH accelerometer for measuring the wear plate vibration with output data rate of 5 kHz. The accelerometer is calibrated to interrupt the Microcontroller Unit (MCU) when acceleration value exceeds the configured threshold value. The MCU enables the FIFO buffer of the accelerometer and sets it on stream mode after receiving inertial wake-up signal. The FIFO buffer needs only to be read after it has been filled, which decreases the load of the processor. In the normal mode, output registers of the accelerometer has to be read faster than set ODR. Reading utilizes the SPI bus. For minimal time consumption, reading is done after buffer is filled. Accelerometer gives an interrupt signal to the controller to start the reading operation. Everything must be done before values in the buffer are overwritten. In 5 kHz case, time window duration is 1/5 000 seconds. After buffer is received, values are safely stored locally on the MCU. Buffer is automatically cleared after reading it and new values can be stored safely. The procedure is repeated several times to gather enough samples for the Fast Fourier Transform (FFT). 250 samples are needed to achieve 20 Hz resolution (2). Sample length is rounded to 256 for convenience and required samples are gathered by reading the buffer 8 times. Each sample has a size of one octet. The sample length 256 is also good for the Bluetooth Sensor since the information can be stored in one attribute slot of the attribute database. Storing 512 and 1024 datapoints will be possible in the future versions of the sensor.

Description of the BLE sensor Sensor contains battery service and custom acceleration service. Battery service contains battery level characteristic [28] which can be only read. The value of the battery level is given in unsigned 8-bit integer and it can have value from 0 to 100. Zero indicates empty battery and 100 full charge.

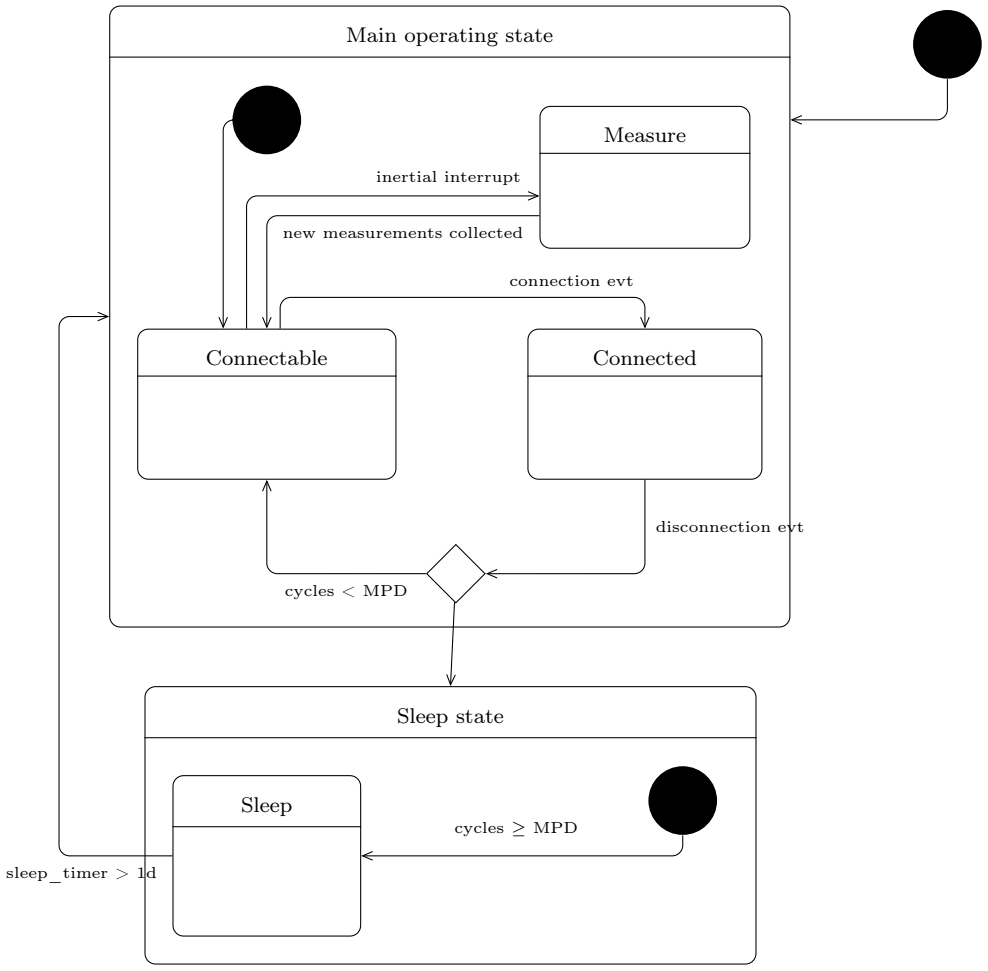

Figure 4.2: Bluetooth Low Energy sensor operational state machine

States The Figure 4.2 represents the functional state diagram of the Bluetooth Low Energy sensor. Initial super state of the sensor is *Main operation* state. In the main operating state, the initial sub state is the *Connectable* state. In this state, the sensor is advertising and awaits for a stimulus from the accelerometer to start measuring the acceleration samples. The advertisement packets are sent periodically and they contain the device name, address and information whether the sensor has a new measurement. The sensor may be connected in this state.

After receiving an interrupt from the accelerometer, the state changes into *Measure* state. In this state, the sensor will immediately start collecting acceleration

data from the accelerometer. After data collection is ready, the state automatically changes back to connectable state and the sensor starts advertising again. During the state transition, a *new data available* flag is set. While this flag is set, an information is appended to a advertisement packet to indicate that there is unread data available. Closer details of measurement process is described in the common functionality part above.

When client initializes connection to the device, the state of the sensor changes to *Connected*. Client can read the acceleration data or battery level in this state. Moreover, the state allows client to configure the sensor. The configurability of the sensor is described later.

After the connection has been closed, there is a state transmit either back to the measure state or to the sleep state. The decision whether to enter the sleep state is done by evaluating the number of cycles that the sensor has done. In the Figure 4.2, constant *MPD* (Measurements Per Day) dictates how many measurements sensor should perform in a day. Number of cycles refers to the amount of measures in between of sleep states. When the sleep state is entered, the number of cycles is initialized to zero. The sensor sleeps for approximately one day and continues measurements on the next day.

Configurability It is possible to configure the service, which stores the acceleration measurements. Configuration enables the client to change sample frequency, scale and threshold of the sensor. The service contains characteristics for each configuration option. Values of these characteristics are mapped to control registers of the accelerometer. New configuration values are used instantly with the next measurement.

Description of the CoAP sensor This works uses a ready-made sensor by Eistec. The board contains Atmel radio chip for 802.15.4 standard. Sensor has Freescale K60 family ARM Cortex-M4 processor. Eistec also provides support for Contiki operating system. The sensor is running a CoAP server to serve the acceleration data. Server resources also include information about the sensor.

4.1.3 Gateway

The main task of the gateway is to automatically transmit the received data to the cloud service. Gateway discovers nearby sensors and sends requests to cloud to create corresponding virtual sensors. Gateway runs an HTTP server which provides web based user interface for wear plate status visualization and for debugging the data transfer. Gateway gathers information of scanned wear plate sensors and caches them locally. The design of the gateway follows ROA based approach.

Gateway structure Gateway is divided into three modules: the core, the 802.15.4 radio module and the BLE module. The core consists of an HTTP server and routines to automatically forward new data to the cloud. The HTTP server allows the end user to monitor and configure the gateway.

Request uses a unique sensor id to check whether virtual sensor already exists in the cloud before creation to prevent duplicates. Each virtual sensor in cloud contain different channels for raw acceleration data and frequency domain data. Gateway receives the data via BLE or 802.15.4 protocol depending on the sensor type and makes HTTP post request to the cloud service. Request contains the received acceleration data and the unique identifier of the sensor. This way cloud service can analyze and store the data to right virtual sensor.

The HTTP server The HTTP server has a RESTful interface, which enables the user to see all visible sensors, fetch data, and configure them. The REST interface can be easily used with browser based user interface, which the web server provides.

Fundamentally, the information exchange between the end user and gateway is based on requests and responses. It is important to define request formats and server representations so that they fit into conditions of the REST. For example, when user requests a sensor with certain UUID, the server would return JSON formatted representation of the sensor, which looks approximately like definition given in the Table 4.1.

Table 4.1: Transferred representation of the sensor

Value	**Type**	**Description**
rssi	Integer (0..100)	Latest rssi value of the sensor
last_updated	Date	Indicates the latest update time of the sensor
device_id	Id	Physical sensor's unique identifier, cannot be changed
battery_level	Integer (0..100)	Expresses the sensor battery level in last updated frame
estimated_change	Date	Estimate for the next wear plate change date
natural_frequency	Integer (0..2000)	Estimate for the current natural frequency of the plate

4.1.4 Cloud

The cloud and its services are on the highest layer of the wear plate monitoring system architecture. The monitoring system cloud is provided by outsourced company. The cloud includes database for the sensor data and Node-RED environment, which allows developing of additional services. The cloud receives, analyzes and stores the received data to its virtual sensors. Cloud is accessed via versatile Javascript API. This API can be either used on the server or the client side. However, services are only used from the gateway for security reasons. The end user is only able to use

the API indirectly, which removes the misusage of the cloud. Only in mobile phone application version of the design, the user is directly communicating with the cloud, which removes the need for the gateway. Moreover, a 64-bytes long key is needed to use the API. Each key can create virtual sensors to the cloud which are only visible to it. A virtual sensor in cloud contains four channels, which are used to store different types of data. Three of them contain raw acceleration data from x, y and z axes. Every channel has an unique channel identifier. Cloud sensor is created on request by the gateway. The sensor has an attribute which contains the unique device id of the sensor. The identifier links virtual and physical sensors together. The last channel is reserved for natural frequency of the wear plate. Cloud analyzes every set of data that is received from the gateway.

4.2 Implementation

Previous section describes the system design. This section covers the implementation phase for sensor, gateway and cloud analysis services. Not all of the design features were implemented.

4.2.1 Sensor

Implementation of Bluetooth Low Energy Sensor The Bluetooth Low Energy sensor (Figure 4.3) operates on a button cell battery. The sensor is inside a casing which is then attached to the back side of the wear plate. The sensor also has an external antenna to improve the range of the sensor.

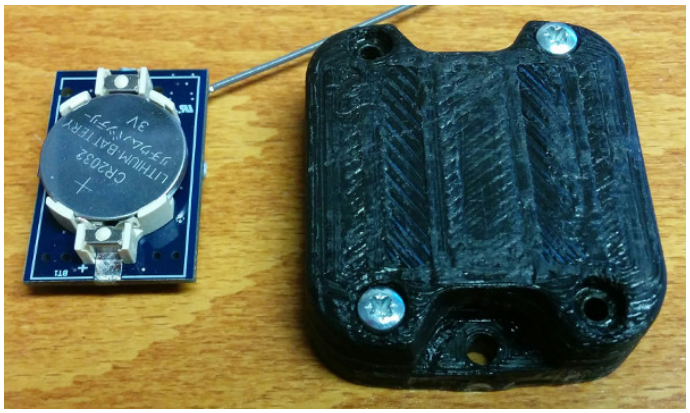

Figure 4.3: BLE sensor and its casing

Implementation of CoAP sensor The sensor board is made by Swedish company Eistec [29], which is also part of the Arrowhead project [30]. Sensor board radiois based on the 802.15.4 standard and it contains the same LIS3DH accelerometer as the Bluetooth Low Energy platform.

Figure 4.4: Top-down view of the wear plate sensor with 802.15.4 compliant radio

The sensor uses Freescale K60 family ARM Cortex-M4 MCU and it is running Contiki OS. Contiki is an open source operating system and it is developed for the Internet of Things. Contiki is a small operating system and it may be used with low-power microcontrollers. Contiki supports IPv4 and IPv6. Furthermore, Contiki supports many recent low-power wireless standards, including the CoAP. Eistec adds the support for Freescale K60 family controllers. The development environment provides a CoAP server, which is configured to serve measured acceleration data.

4.2.2 Gateway

On the physical aspect, Gateway comprises an HTTP server and two radio modules for both Bluetooth Low Energy sensor and CoAP sensor. Gateway is implemented in Node.js environment and to cover the required functionality, it is needed to integrate various Node.js modules to the system. These modules are needed, e.g., to use the CoAP and the BLE.

Required Node.js modules Node.js modules provide the required functionality. Most important of these modules, which are used with the gateway, are listed and their functionality is described shortly. Modules required by the gateway are:

Express.js Express.js is an HTTP framework module for Node.js [31]. With Express.js module, gateway implements the HTTP server, which provides the REST interface and also the Web based user interface.

Noble Noble is a Bluetooth Low energy central module for Node.js [32]. Noble is able to, e.g., scan devices, discover services and characteristics, connect to a device, and read or write to characteristics. The gateway utilizes Noble to interact with BLE sensors.

Coap Using the CoAP in Node.js environment is possible with the node-coap module [33]. After establishing the connection to the physical part of the gateway

802.15.4 module, it is possible to reach the PAN with CoAP requests from Node.js environment. In addition to normal requests, the module has a support for observing resources and for the block transfer.

Cloud interface module Cloud module, which provides the cloud API, is a private module. Library was not originally intended to be used with Node.js but since it is written in javascript, porting the library to Node.js is trivial. Cloud interface is hidden from the End User by placing it to the gateway.

Physical part of the 802.15.4 module The physical receiver of the gateway utilizes the same Eistec sensor board as the sensor but with change of the board firmware. The sensor is connected to separate debug board which enables virtual network interface over a serial connection. Contiki operating system provides a tool called *tunslip* to establish the Serial Line Internet Protocol (SLIP) tunnel between physical a serial port and the virtual network [34]. The physical part of the 802.15.4 module enables the gateway to make CoAP requests to the sensor network in Node.js environment.

Physical part of the Bluetooth central device module A common Bluetooth dongle functions as a central device on the gateway. The dongle supports the 4.0 core definition, which describes the Bluetooth Low Energy. This dongle is utilized with Noble module.

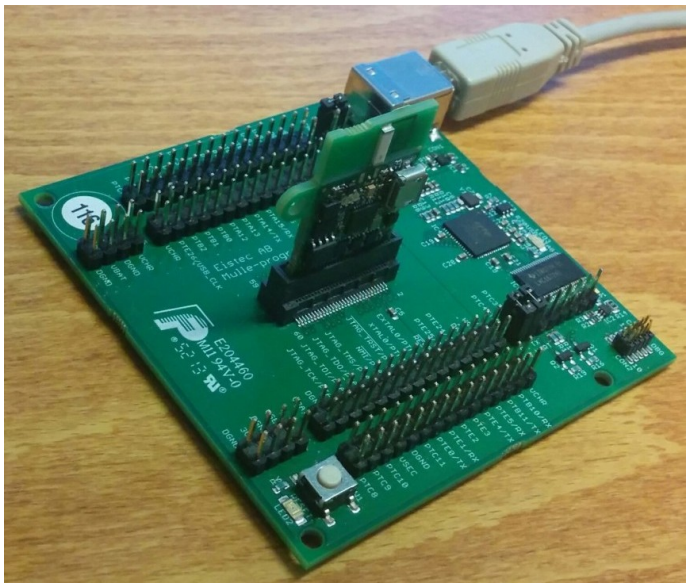

Figure 4.5: Node requests and receives data from sensor nodes and delivers the messages to the gateway

4.2.3 Cloud

The data analysis module is implemented in Node-RED environment. The analysis of the acceleration signal is written in R code [6]. An additional node is needed to support the execution of the R code in the environment. The user can require lifetime estimate for certain wear plate from the cloud service.

In analysis (Figure 4.6), the acceleration signal is received via TCP port. The signal is converted into JSON format and then converted into frequency domain with a block that executes R code. The conversion to the time domain utilizes DFT introduced in section 2.3.2. In the R code block, the natural frequency is calculated by determining the maximum amplitude of the frequency domain presentation of the signal. The result is capsulated with a time stamp and other characteristics of the acceleration signal. The results is sent back via the same TCP socket.

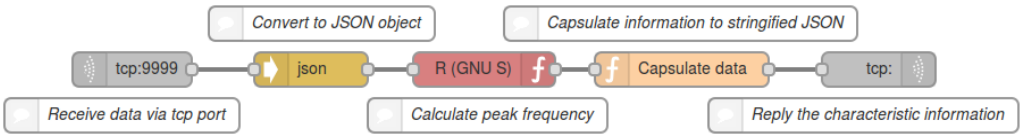

Figure 4.6: Service in cloud for calculating natural frequency. Implementation is done in Node-RED environment

Lifetime estimation service fetches the data from frequency channel of the desired sensor. A simple linear model is fitted into the data which is then used to extrapolate an estimate for the lifetime. Usually dataset contains several error frequencies that are caused by either misinterpretation of the acceleration signal or poor quality of it. Low quality signal contains no clear natural frequency, which may be caused by many stones hitting the surface at the same time. These frequencies are clearly separated from point scatter and they cause error in linear regression model. Therefore, they are removed from the dataset when estimating the wear plate lifetime. Removal of these points is done by first fitting linear model to all points, then calculating the Mean Squared Error (MSE) of the data points and then removing points which squared error clearly exceeds the calculated MSE value. After removing unreliable samples from the set, the linear model is fitted again with reduced data which usually leads to a better estimate of the wear plate lifetime. The data may still contain errorneus frequencies that were not removed due the high MSE value caused by already removed points. Error of these removed points may easily exceed 500 Hz due to misinterpretation and points with smaller, but still considerable, error values are left in the dataset. Reducing procedure can be repeated for remaining samples to improve the final result. In the future it is desirable not to store measurements that are not valid. However, validating a measurement is a difficult challenge.

4.2.4 Implementation notes

The workload of this work divides into three layers: sensor layer, gateway layer and cloud layer. This section further clarifies what was actually implemented on these

three different layers and what technology was already offered for the use.

Sensor layer On the sensor level, I studied two different sensor platforms. Firstly, I researched 802.15.4 based sensor platform with already given hardware platform and software development environment. The development environment enabled me to quickly conduct acceleration measurements and send them to the PC gateway over the CoAP protocol. I was also given tools to create "physical" CoAP gateway between PC and sensors. The technology was provided by Eistec.

Gateway layer I implemented the gateway with Node.js and Beaglebone. I developed a gateway application that contains an HTTP server and a Bluetooth Low Energy module, which read the data from the sensor and cached it locally on the gateway. The gateway is also capable to store the data in the cloud.

Cloud layer The cloud framework was provided by third party. The cloud provided me a framework to develop services to analyze the data in Node-RED environment. In Node-RED, I could also use R code. The cloud also provided me an interface for storing the data.

5 Monitoring system use cases

This section demonstrates the monitoring system that was developed during this work. The section concentrates separately on each different layer and introduces their functionality.

5.1 Sensor

5.1.1 Bluetooth Low Energy

Use case: discover battery service and read battery level characteristic Next use case example inspects the Bluetooth Low Energy traffic, when the sensor services are discovered and the battery service characteristic value is being read. Bluetooth Low Energy traffic is monitored with Wireshark [35] and the example includes screen captures of the program.

First the central device requests the Bluetooth Low Energy sensor to list all services it contain via GATT. From these services, the battery service is further inspected. Battery service is specified in Bluetooth standard and service identifier 0x180f is associated with it. Figure 5.1 shows the received packets where discovered services are introduced. The battery service is highlighted in the Figure. The Handle value of the service defines its location in the attribute server. The Group End Handle determines the range of the service, i.e., how many attributes belong to the service. Finally, the value 0x0f18 tells the type of the service. Bluetooth packets utilize the little byte endianness [7, vol. 3, p. 537], which is 0x180f when represented in big endian format (the battery service identifier).

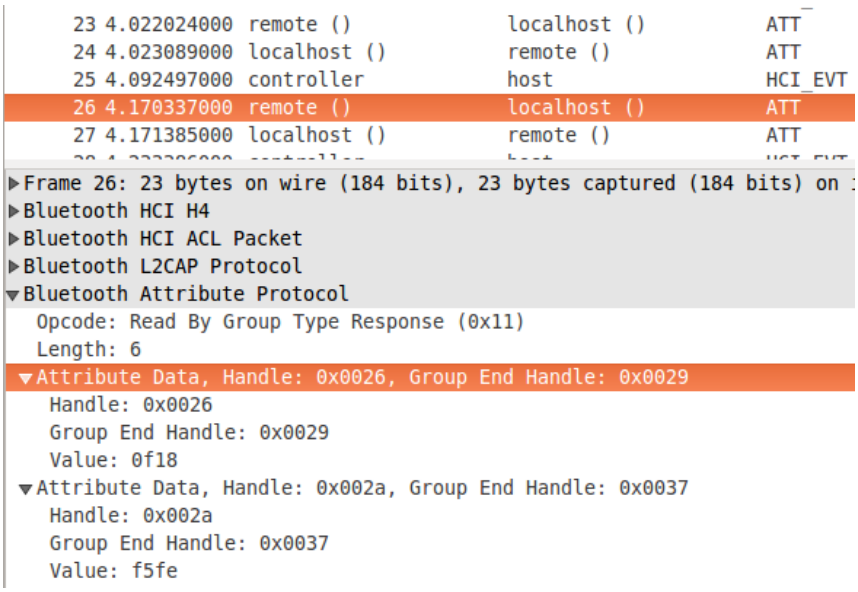

Figure 5.1: Wireshark shows the Bluetooth Low Energy traffic. Battery service is located on the remote peripheral.

The characteristics of the battery service can be discovered after the service itself has been discovered. Characteristic discovering utilizes ATT *Read By Type* request that is configured to only read attributes with characteristic type. Then, all the characteristics of the service are returned. The battery service has only one characteristic, which indicates the battery level. The Figure 5.2 shows the sole characteristic of the battery service. The Handle with value 0x0027 is the location of the characteristic in the attribute database. The value 0x12 28 00 19 2a is the value of the characteristic (in little endian). Conversion to big endian format gives the value 0x12 00 28 2a 19. Grouping of the fields must remain the same after the conversion, thus conversion between endianness must be done in groups. First octet 0x12 defines the properties of the characteristic. Two second octets 0x00 28 define the location of the characteristic value. The characteristic value is where the actual battery level value is stored. Last two octets 0x2a 19 tell the uuid of the characteristic. Characteristic with uuid 0x2a 19 is specified in battery service and it is the battery level characteristic.

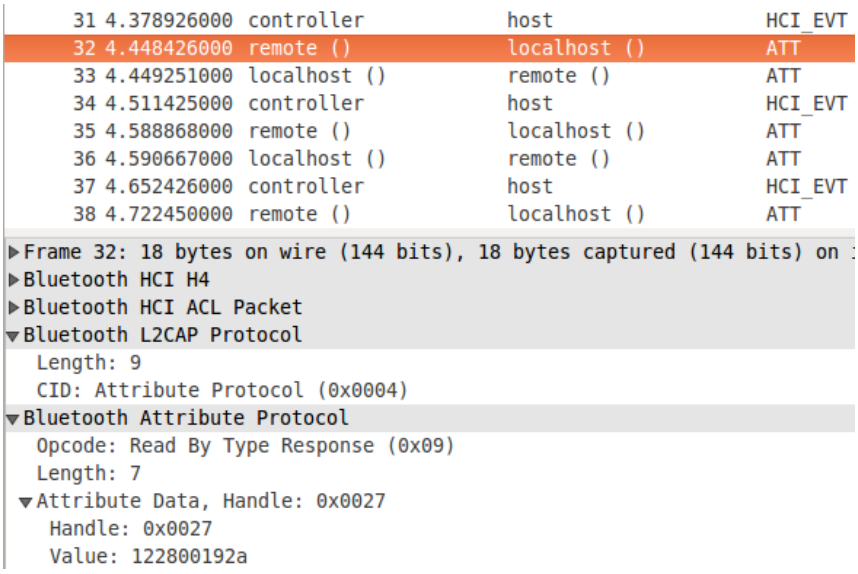

Figure 5.2: Battery service characteristic

Read request reads the current value of the battery characteristic. Value field in the response packet of the characteristic is the current battery level. The current level of the battery is 0x55, or 85 in decimal notation (Figure 5.3). The decimal value is the value of the maximum battery level in percents.

In conclusion, this use case demonstrated the functionality of the battery service and fetched the value of the battery level characteristic. This was done in three steps:

1. Discover all services of the device.
2. Identify the battery service and discover all characteristics of it. By definition, the battery service has only one characteristic, which indicates the battery

level.

3. Read the value of the battery characteristic. The value is contained on other attribute in the attribute database.

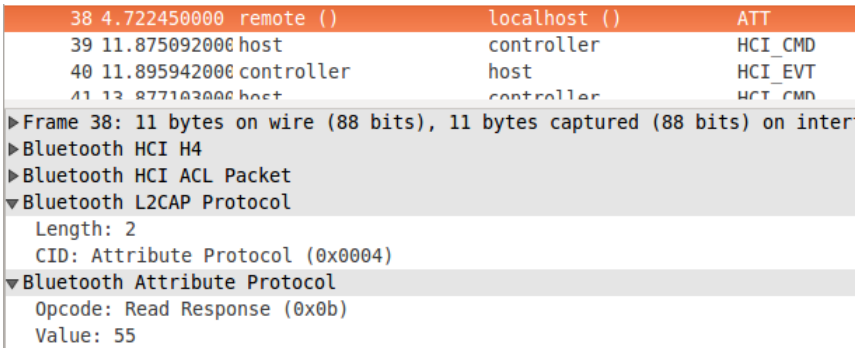

Figure 5.3: Battery service characteristic

Use case: discover acceleration service and read value of service's acceleration characteristic In this use case, a BLE central device is used to discover the acceleration service and read the value of the acceleration characteristic, which contains the acceleration data. The data comprises 256 octets. In this use case example, data is not gathered with accelerometer but it is generated on the sensor. This use case contains more phases than previous use case and there is no detailed explanation about PDU exchange.

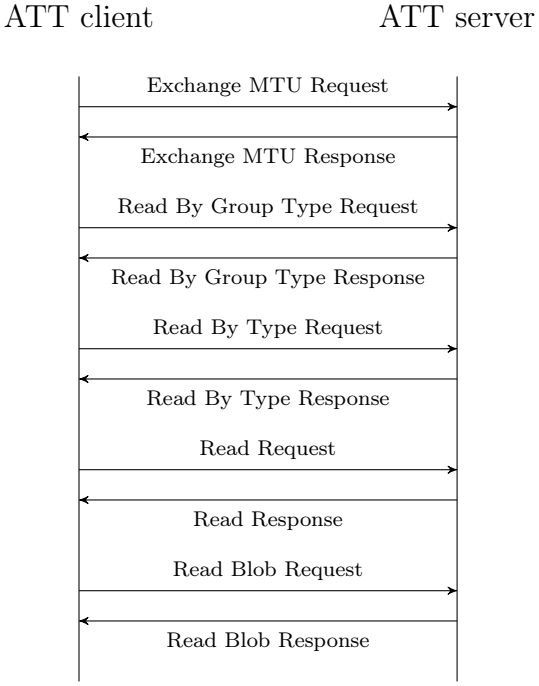

Figure 5.4: Sequence diagram for reading acceleration characteristic value

Packet exchange of the given use case is depicted in the Figure 5.4. First the client requests for bigger MTU packet size for efficiency. Maximum size of the MTU is 256 and if accepted by both ends, the server can send all the data at once without the need of multiple consecutive read blob requests to complete the transfer. This reduces the amount of transmitted packets, which also improves the energy efficiency. If the client receives confirmation to the exchange MTU request from the server, the allowed packet size is bigger than by default.

Like in the previous use case, the next phase is the service discovery. The explanation of this procedure is omitted here because it is similar with previous use case. The only difference is that more packets are exchanged because all the services of the sensor were enabled, which required several requests to fetch all the primary services. After discovering the acceleration service and corresponding characteristics, the central device reads the value attribute. If bigger value for the MTU is not accepted, the central device needs to make multiple read blob requests.

5.1.2 CoAP sensor

Use case: discover server resources Service discovery procedure of CoAP server is discussed in the section 3.3.1. This use case demonstrates the service discovery of the sensor using the predefined URI /.well-known/core. In this use case, the sensor contains only one resource that is located in */acceleration*. However, the final version of the sensor would have numerous services for, e.g., reading the battery level and for the sensor configuration. Requesting the acceleration resource returns the latest measurement of vibration.

First the client (gateway) generates and sends a Confirmable message with Uri-Path option dictating the resource location to the sensor. The server on the sensor receives the message and replies with Acknowledgement message. The Acknowledgement has an option Block2, which means that the transfer is done in blocks. Block transfer enables the transmission of bigger data chunks in separate blocks. The information of available resources is stored in the message payload. The Block2 option contains a *More* flag to indicate that not all of the data could be transferred in one block and the client should make a new request to fetch the remaining data. The client must append a new option to the request which defines the next block of the desired resource. In the last Acknowledgement packet from server, More flag is unset to inform the client. The discovering procedure has now informed the client about all the available services. The client can store the interface description.

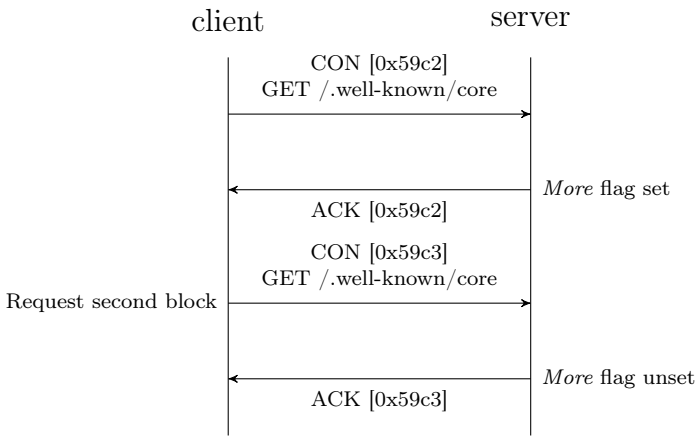

Figure 5.5: Discovering server resources by CoRE

Use case: make CoAP request to /acceleration resource This use case demonstrates the message interchange when requesting the */acceleration* resource from the sensor. The transaction starts with the client generating a Confirmable request. The client sets the *Uri-Path* option to */acceleration*. Moreover, the *Block2* option is set to enable the transfer in data chunks. The client sends the request to the sensor, which responds with ACK message. Respond also contains Block2 option with *More* flag set to inform the client that there will be more packets to be

received. The Block2 option also contains the *Block Number* parameter that is an index number for the packets starting from zero. In this use case, the sensor sends two more packets to the server. In the last packet, the More flag is unset and and Block Number value is 2. Figure 5.6 shows the message interchange.

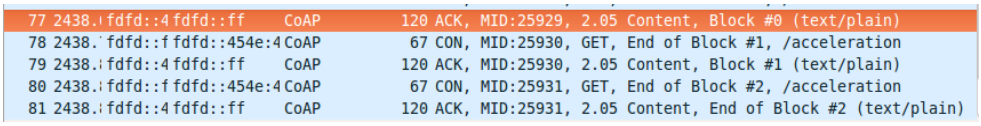

Figure 5.6: A screen capture of Wireshark showing the CoAP traffic while requesting */acceleration* resource from the sensor

5.2 Gateway

The gateway is responsible of discovering BLE and CoAP sensors. The gateway caches information about each discovered sensor. This information includes sensor identifier, measured data and other available information (Table 4.1). All the information about sensors is available via the gateway REST interface. Moreover, the gateway serves a browser based user interface on its HTTP server for debugging the interface and for monitoring purposes. In the user interface, the user can see all the discovered sensors and information about them. In addition, the user can test data analysis with the interface.

Use case: receive and cache sensor data In order to cache the data from the sensor, the gateway must first receive it from the sensor. Since there are two types of sensors, the use case has to be described for both sensors. These use cases link with previously handled use cases. Because of previous use cases also demonstrated how the central device fetches the data, this use case will not describe it again. However, now the focus is more on the gateway side.

BLE sensor To read a characteristic that contains the measured sensor data on the Bluetooth sensor, the gateway must first establish connection to the sensor. After the connection has been established, the gateway can perform GATT discovering procedures and read the collected sensor data. It is also possible to read the data characteristic without the need of discovering services and characteristics, if the location of the characteristic is known in attribute database. The gateway utilizes the Noble module for operating the Bluetooth dongle. After the reading operation has been completed, the gateway stores the results in a javascript object called *devices*. The object is comprises key-value pairs. The key is the device uuid and the value is another javascript object that contains the data that the gateway received from the sensor: battery level, acceleration data and rssi value. The gateway also attaches timestamp to this object.

CoAP This use case is related to the previously mentioned use case where the */acceleration* resource is requested by the client. The gateway automates the process and generates requests with specified interval. Currently the gateway is not able to subscribe to a resource, which would be more practical way of receiving the data from the sensor. When observing the resources (p. 31), sensor would send new data automatically to the gateway. Then there is no need for unnecessary polling messages.

After the gateway has generated and sent the request to the sensor, the sensor sends a response. The gateway then attaches the result to the same javascript object as in the previous use case. However, the key is now the IPv6 based address of the CoAP sensor. Battery level and rssi value are not yet implemented for the CoAP sensor and thus they are not stored in the javascript object.

Use case: fetch cached data from the gateway The previous use case describes how the gateway caches the sensor data. After the data has been correctly stored, the client can fetch it with a simple HTTP request with the REST interface. To receive an overview of a sensor, the client can make a request to a following URI: */sensors/ble/:uuid/data*. The server returns the rssi, the battery level and the timestamp when the sensor was updated.

The cached data is addressed with following URI: */sensors/ble/:uuid/data*. For example, data from BLE sensor with address *80:ea:ca:00:00:03* would be located in */sensors/ble/80eaca000003/data*. The client makes an HTTP request using the GET method and the server responds with an array containing the previously fetched sensor data in human readable form. The HTTP status message of the response is 200, which indicates successful transaction. The response is delivered in JSON format. If gateway does not have cached data from a certain device, then the server will response with status code 404 Not Found.

Figure 5.7 shows a request that fetches the cached data from the gateway. In this example, the data is received from a real physical BLE sensor. The data is then plotted in the Web UI (Figure 5.8). In this case, the sensor measured only noise since the threshold was set to zero. The negative bias is caused by the gravity.

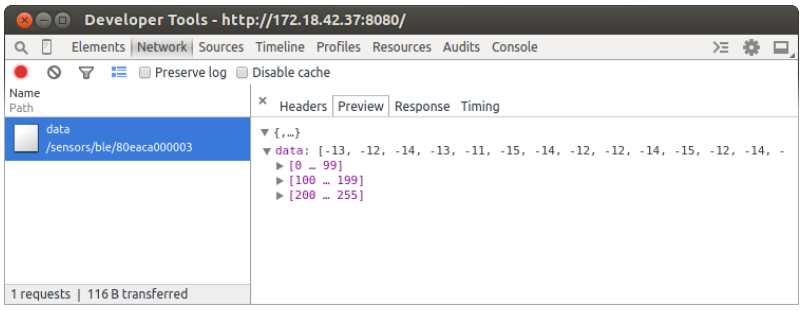

Figure 5.7: Developer tools shows a request that is made from Web UI to a sensor. Server response returns real measurement data to client in JSON format

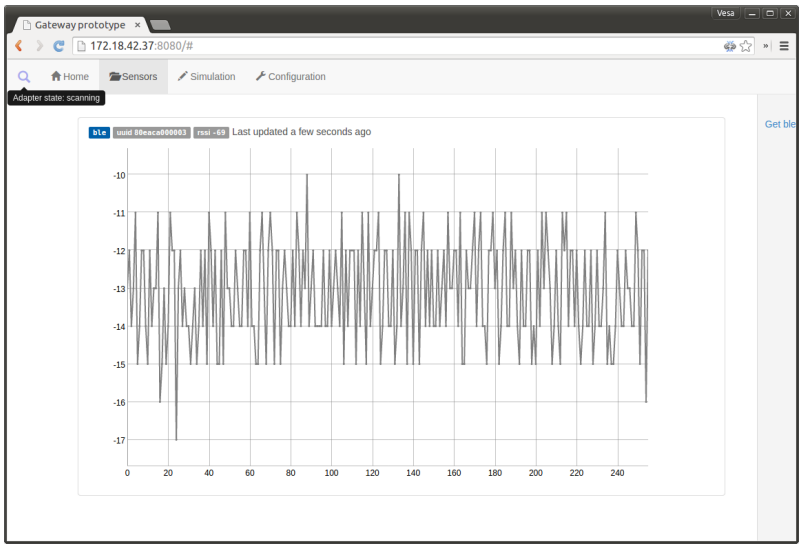

Figure 5.8: The cached data can be inspected in the gateway Web UI

The use case is demonstrated only on BLE sensor, but the use case would be the same for the CoAP sensor since the gateway will cache the data in same format for both sensors. Only the URI would change. This feature is still to be implemented in the future version.

Use case: analyze simulation data This use case describes the action performed on the gateway when sending data to the cloud service for analysis. The gateway user interface provides simulation environment which allows the user to generate simulation data and send it to the cloud.

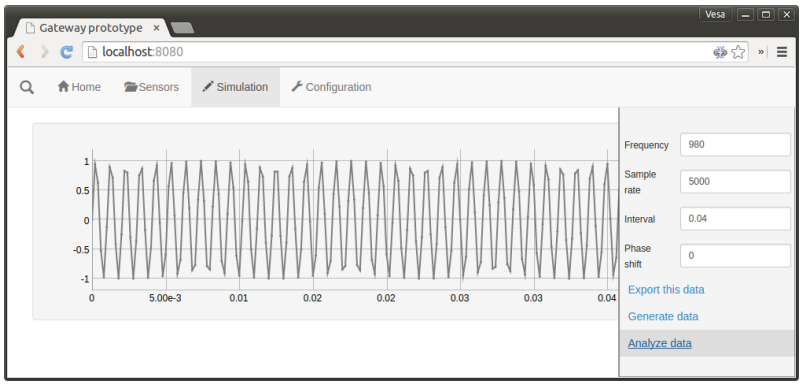

Figure 5.9: Gateway user interface provides simple simulation capabilities for testing the analysis service of cloud interface

In this use case test, frequency of the sine wave is set to 980 with a sample rate of 5 kHz, which is also the sample rate of the used LIS3DH accelerometer. Samples are

collected for 0.04 seconds, which equals to approximately 200 samples. Simulation interface is depicted in the Figure 5.9.

The generate data button will generate data defined by the set parameters and store it locally to the end user. Clicking the Analyze data will send the stored data to the gateway which passes the request to the cloud. The cloud analyzes the data and responds to the gateway, which responds to the end user with analysis information (Figure 5.10). The analysis contains the natural frequency of the signal, length, resolution and the sample rate that the analysis used.

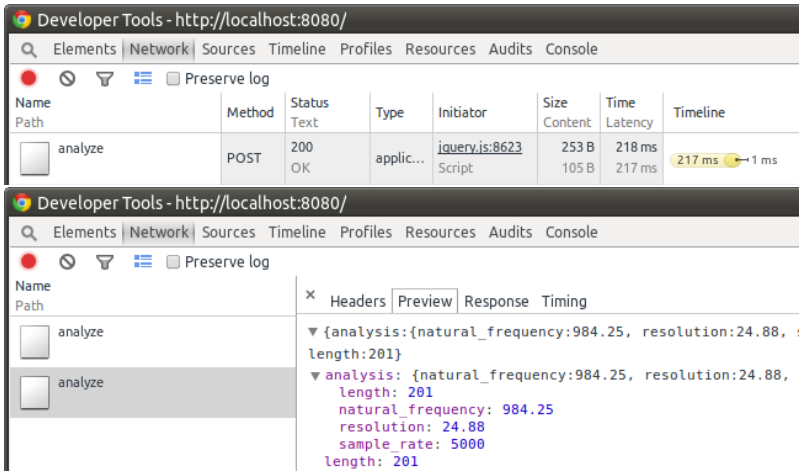

Figure 5.10: Analysis request sent to the gateway, which forwards it to the cloud. Cloud responds with information about the data

6 On site measurements

This section shows the final measurements in Kemi chromium mine. The measurements are only collected from two days and the frequency drift cannot be seen. However, these results encourage to develop the system further. The measured spectrum has expected dominant frequency components.

6.1 Hammer generated impact

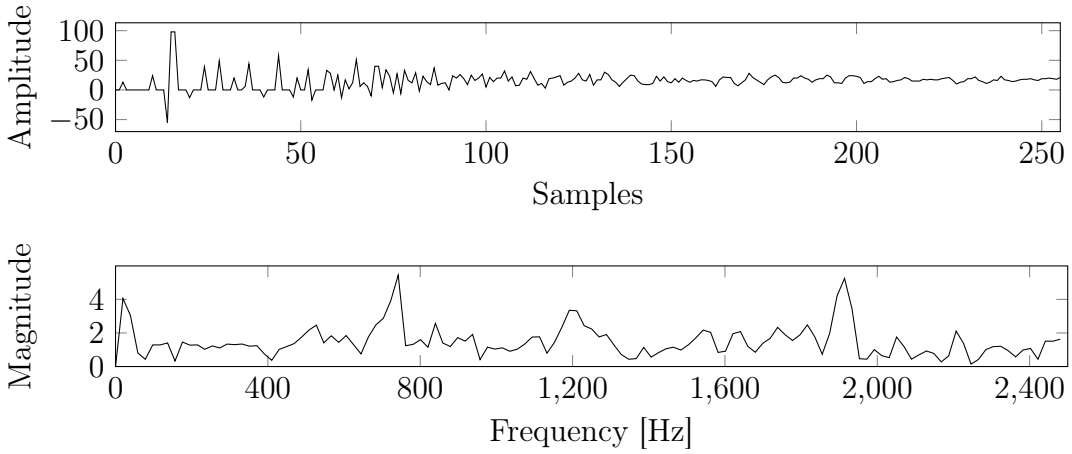

Figure 6.1: First measurement with hammer

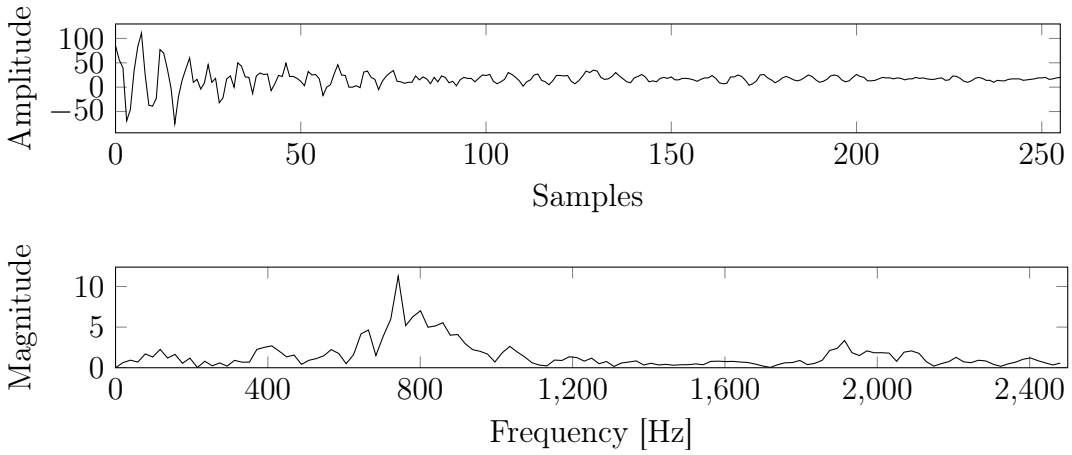

Figure 6.2: Second measurement with hammer

In Figures 6.1 and 6.2, the data is gathered from a wear plate of the conveyor feed. Figures show the collected raw data and corresponding frequency domain representation. The sample rate of the sensor is 5 kHz. The process is not running

in these samples but instead the wear plate is hit with a hammer. The wear plate starts vibrating and the data is then collected with the sensor. The sensor triggers measurement automatically when a specified acceleration level is exceeded. In both samples, there is a frequency peak in a bit under 800 Hz, which should be approximately correct frequency for a new plate.

6.2 Real process measurements

Figures 6.3 and 6.4 show real data when the actual ore is running through the process (Figure 2.3). It is noticeable that there is more noise in the measurements, which is seen in the frequency domain. There are also much more higher frequencies that lowers the quality of the measurements. Higher frequencies tend to dampen faster than the natural frequency. Therefore, it would be a good idea to delay the start of the measurement for some milliseconds. In the second measurement (Figure 6.4), the sensor is clearly triggered from too small impact. Natural frequency is not visible in the frequency domain view.

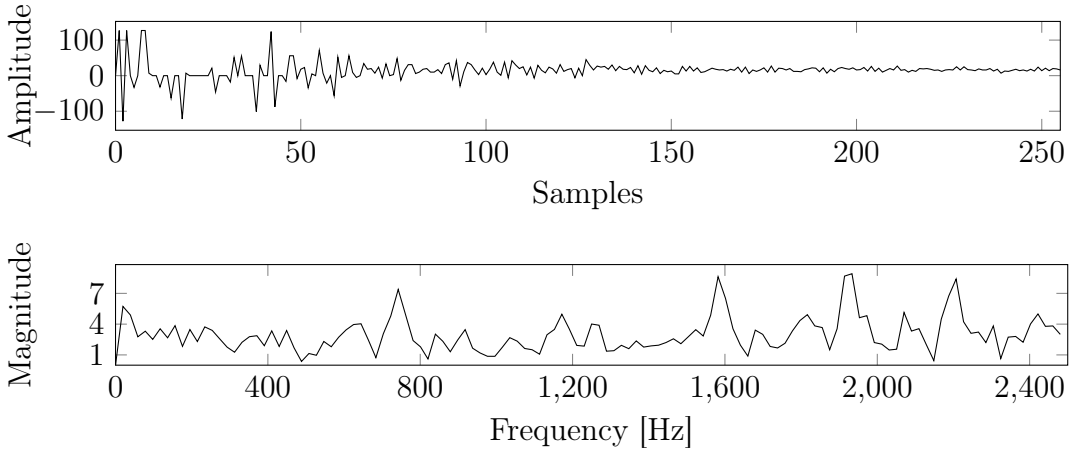

Figure 6.3: Real measurement with actual process

6.3 Quality of the measurements

It is promising that the same natural frequency, which was measured with the hammer experiment, is also visible in real measurements. Unfortunately, it is typical that the sensor is triggered by too small impact, in which case the gathered sample will contain high amount of noise. In shown measurements, the impact may be too strong (Figure 6.3). The sensor is set in 8g mode. The sensor digitalizes measurements to a 8-bit register. Then, 127 denotes 8g and -127 -8g level. It is clear that some measurements are saturated in the beginning of the measurement.

Changing the sensor scale to maximum 16g would resolve saturation issues. It would also allow using bigger threshold value. However, if measurements are conducted with high g-level, then the interesting part, where natural frequency is clearly

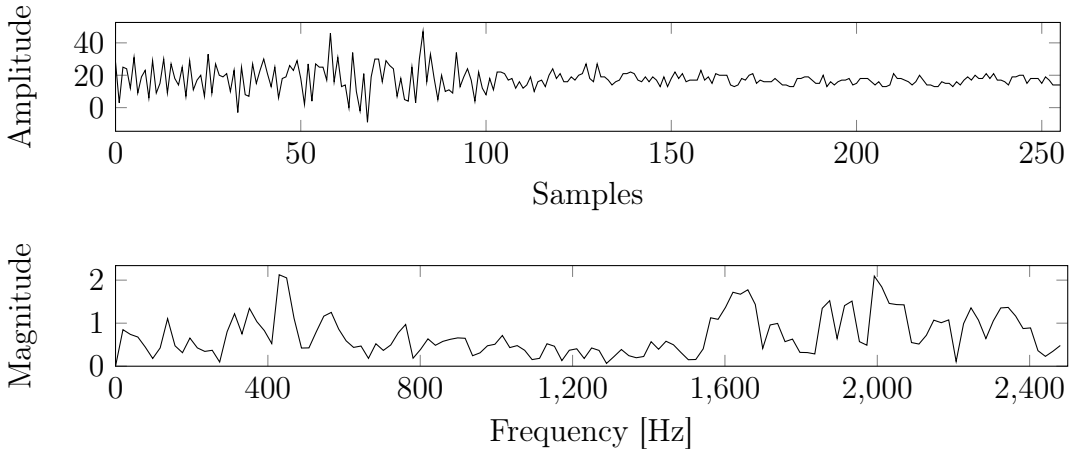

Figure 6.4: Second measurement with actual process

visible, will lose resolution. The latter part of the measurement does not have as high amplitude as the beginning of the signal. Lower amplitude part of the signal is mapped closed to zero value if the whole measurement is acquired with same g-level. In other words, most of the range is unused due to the too high scale setting. One solution would be to change the scale on the fly. However, complex sensor calibration would require a lot of testing and desing work.

What makes the problem more challenging in the real environment is that there may not be one single impact. The dropped ore is composed of many different size pieces of rocks. These pieces may hit the plate simultaneously and cause unwanted results.

7 Conclusions

7.1 Review

Automated condition monitoring aims to reduce maintenance breaks and shorten downtimes. Shorter downtimes of the process improve productivity and efficiency. This thesis work has conducted preliminary studies concerning the industrial aspect of the Internet of Things (IoT). The IoT will be one of the most rapidly growing areas in the near future [36]. The work studied the Bluetooth Low Energy and the CoAP protocol placed on top of the 802.15.4 standard and their suitability for wireless industrial communication. The results of these two different technologies are very promising in low data rate wireless sensor networks and they are both very potential future technologies in the industrial usage. However, the Bluetooth Core is more complete compared to relatively lightweight 802.15.4 standard, which however is constantly emerging and will have its place in the world of the IoT.

This work introduced a three level condition monitoring system. Lowest level consists of wireless sensors that are connected to the gateway. The gateway gathers data from these sensors and stores it to the cloud. The system was tested with simultaneously with seven sensors. The cloud and its analysis services become essential building blocks of the system especially when the data volume grows higher [37]. The overall results of the prototype system are encouraging. Such condition monitoring systems may be taken into real use in less than 5 years. However, still many used technologies are under development and they are constantly bringing more stability and features.

Comparison of BLE and CoAP sensors The Bluetooth Low Energy sensor structures its functionality in two services, which provide acceleration data and information about battery level. Bluetooth Low Energy utilizes frequency hopping and it is very tolerant againts disturbance. The BLE works in unlicensed 2.4 giga hertz frequency. Because the range is not licensed for the Bluetooth, the transmission power must be kept very low. In addition, the used frequencies are highly absorbant [13] and the environment has a great effect to the stability of the network.

Other sensor is based on the IEEE 802.15.4 standard and it utilizes the CoAP protocol. Unlike the Bluetooth, the CoAP protocol is connectionless and it is based on requests and responses. The current version of the gateway is only able to poll sensors in the network with CoAP requests which causes unnecessary network load. However, Erbium CoAP server, which the sensor utilizes, supports resource observing [19]. Subscribing to the acceleration resource would split the network load and remove unnecessary traffic caused by polling.

Power consumption The Bluetooth sensor operates on a button cell battery, which has a very limited amount of energy stored in it. Normal CR2032 type battery typically contains about 230 mAh and the sensor must operate at least several months with it. Due to the energy constraints of the button cell battery, the power consumption is strictly regulated and the BLE sensor is kept in sleep mode

whenever possible. Moreover, transmitting data over the air is expensive in terms of energy. Therefore, transmitted packets are kept in minimum and they are as short as possible. The device advertising interval is kept long to further minimize the radio usage.

7.2 Further development

The current prototype system is able to measure vibration on the sensor level and transfer it further the gateway. The gateway is able to cache this data and send it to the cloud for analysis service.

The stability of the gateway must be improved. There is not enough of exception handling. Currently gateway may fall in an inoperable state due to the Bluetooth adapter failure. Sensor configuration is not yet possible with the gateway. The calibration is done with phone BLE debug application by manually writing hex values to characteristics. The HTTP interface defined in Appendix A is not final. The interface is constantly changing to meet the requirements of the emerging system.

There are also many mechanical problems to overcome. The sensor is difficult to attach to the wear plate. Wear plate itself is made of exremely hard material and therefore screws cannot be utilized. Magnetic attachment may affect the measurement too much. Currently the sensor is glued onto the surface of the plate with satisfying results. The surrounding environment is very hostile to the sensor on the target site. The room is filled with fine stone dust which will break the sensor with time if the sensor is not encased properly. Moreover, the aggressive tremor is capable of destroying the sensor by breaking components or connections on the board.

This work was unable to measure the long term frequency shift of a wear plate due to the stability issues of the sensor. However, an satisfactory level of completeness was reached with the prototype system. In the future the sensor will be tested more in real environment.

7.2.1 Expandability of the system

Automated condition monitoring that utilizes low energy wireless sensors have numerous application areas in quarrying and mining that this thesis does not cover. In addition to wear plates that are located in the conveyor feed, similar acceleration sensors may be installed into, e.g., ore screeners. Certain stone crushers include a buffer bin, which is filled with stones. The resonance frequency of the buffer walls vary depending on the fill level. This phenomenon can be compared with a ruler placed on the edge of a desk. The ruler starts resonating when it is bent and released from its free end. The frequency of the resonation depends on how far the ruler comes from the table. Hence, vibration analysis may provide an alternate method for monitoring the load level of the buffer bin.

Other feasible application areas may include condition monitoring of wear parts in mobile crushing stations and other various crushers (gyratory and cone crushers).

7.2.2 Emerging technologies

As mentioned, the IoT is currently extremely fast growing area on many fields, including industrial networking. Many various standards are competing to gain the dominating position. This section reviews Bluetooth Core 4.2 definition and upcoming 802.11ah standard. There also exist various other quickly emerging standards.

Bluetooth Core 4.2 The newest version of the Bluetooth Core definition was published in the end of the year 2014. The version 4.2 aims to make the Bluetooth Low Energy ideal for the usage of the IoT. It introduces the latest version of the Internet Protocol (IPv6) to Bluetooth Low Energy devices [38]. It brings the Internet and Bluetooth devices closer to each other. In very near future, Bluetooth Low Energy sensors are truly able to, e.g., communicate with with servers in the Internet via REST interfaces. New core version also improves speed, security and power efficiency.

802.11ah The WiFi standard 802.11ah enables the usages of WiFi in sub 1 GHz frequencies [39]. The standard adds the support of WiFi to embedded devices. Depending on the energy competitiveness of the upcoming standard, it may achieve a significant position among other already existing wireless communication standards and definitions. With the 802.11ah standard, sensors would be able to connect to the same network and directly communicate with the gateway. There would not be need for any extra protocols. However, it is doubtful that the upcoming WiFi standard could compete in low power communication with currently existing standards.

References

[1] Metso. *Services: Supervision and technical support*. 2014. URL: http://metso.com/services/field-services/supervision-and-technical-support/.

[2] Princeton University. *Wordnet, quarrying*. Oct. 31, 2014. URL: http://wordnetweb.princeton.edu/.

[3] Metso. *Crushing And Screening Handbook*. Ed. by Keijo Viilo. 2011.

[4] Hardox. *HARDOX homepage*. 2014. URL: http://www.hardox.com/.

[5] Julius O. Smith III. *Mathematics of the Discrete Fourier Transform (DFT)*. BookSurge Publishing, 2007. ISBN: 978-0-0745607-4-8.

[6] Brett Lantz. *Machine Learning With R*. Packt Publishing, 2013. ISBN: 978-1-78216-214-8.

[7] Bluetooth SIG. *The Bluetooth Specification Core Version 4.1*. 2013. URL: https://www.bluetooth.org/en-us/specification/adopted-specifications.

[8] IEEE Computer Society. *Std. 802.15.4-2011: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. 2011.

[9] ST Microelectronics. *LIS3DH: MEMS digital output motion sensor ultra low-power high performance 3-axis "nano" accelerometer*. AN3308. 2011. URL: http://www.st.com/web/en/resource/technical/document/application_note/CD00290365.pdf.

[10] Roy Want, Bill Schilit, and Dominik Laskowski. "Bluetooth LE Finds Its Niche". In: *IEEE Pervasive Computing* 12.4 (2013), pp. 12–16. ISSN: 1536-1268.

[11] Nordic Semicondutors. *Bluetooth low energy wireless technology backgrounder*. 2010. URL: https://www.nordicsemi.com/eng/News/Press-Center/Press-Backgrounders/Bluetooth-low-energy-wireless-technology-backgrounder.

[12] Carles Gomez, Joaquim Oller, and Josep Paradells. "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology". In: *Sensors* 12.9 (2012), pp. 11734–11753.

[13] Robin Heydon. *Bluetooth Low Energy: The Developer's Handbook*. Pearson Education, 2012. ISBN: 978-0-13-288836-3.

[14] Atmel. *IEEE 802.15.4 MAC User Guide*. 2006. URL: http://www.atmel.com/Images/doc5182.pdf.

[15] Internet Engineering Task Force. *RFC 7230: HTTP/1.1 Message Syntax and Routing*. 2014. URL: https://tools.ietf.org/html/rfc7230.

[16] Internet Engineering Task Force. *RFC 7231: HTTP/1.1 Semantics and Content*. 2014. URL: https://tools.ietf.org/html/rfc7231.

[17] HTTPbis Working Group. *Hypertext Transfer Protocol version 2*. 2015. URL: https://http2.github.io/http2-spec/.

[18] Internet Engineering Task Force. *RFC 7252: The Constrained Application Protocol (CoAP)*. 2014. URL: https://tools.ietf.org/html/rfc7252.

[19] CoRE Working Group Klaus Hartke. *draft-ietf-core-observe-14: Observing Resources in CoAP*. 2014. URL: https://tools.ietf.org/html/draft-ietf-core-observe-14.

[20] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN: 978-0-596-52926-0.

[21] Roy Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[22] Internet Engineering Task Force. *RFC 6690: Constrained RESTful Environments (CoRE) Link Format*. 2014. URL: http://tools.ietf.org/html/rfc6690.

[23] NodeJS. *Node.js homepage*. 2014. URL: www.nodejs.org.

[24] Pedro Teixeira. *Professional Node.js: Building Javascript Based Scalable Software*. John Wiley & Sons, Inc., 2012. ISBN: 978-1-118-18546-9.

[25] Mike Cantelon et al. *Node.js in Action*. Manning Publications, 2014. ISBN: 978-1617290572.

[26] IBM Emerging Technology. *Node-RED*. 24 2, 2014. URL: http://nodered.org/.

[27] BeagleBoard.org Foundation. *Beagle Board homepage*. Feb. 24, 2015. URL: http://beagleboard.org/.

[28] Bluetooth SIG. *Bluetooth Developer Portal. Gatt Specifications, Characteristics*. Oct. 7, 2014. URL: https://developer.bluetooth.org.

[29] Eistec AB. *Home page*. Sept. 30, 2014. URL: http://www.eistec.se/.

[30] Arrowhead. *Partners | Arrowhead*. Sept. 30, 2014. URL: http://www.arrowhead.eu/partners/.

[31] Express.js. *Express.js homepage*. 2014. URL: http://expressjs.com/.

[32] Sandeep Mistry. *Github: A node.js BLE (Bluetooth low energy) central module*. Oct. 17, 2014. URL: https://github.com/sandeepmistry/noble.

[33] Matteo Collina. *Github: node-coap*. Oct. 1, 2014. URL: https://github.com/mcollina/node-coap.

[34] Contiki OS. *Github: contiki-os*. Sept. 30, 2014. URL: https://github.com/contiki-os/contiki.

[35] Wireshark Foundation. *Wireshark*. Oct. 10, 2014. URL: https://www.wireshark.org/.

[36] Kevin Ashton. “That ‘internet of things’ thing”. In: *RFiD Journal* 22 (2009), pp. 97–114.

[37] René Hummen et al. “A Cloud design for user-controlled storage and processing of sensor data”. In: *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2012, pp. 232–240.

[38] Bluetooth SIG. *Bluetooth Core Specification 4.2, Quick Reference Guide*. Feb. 6, 2015. URL: `https://www.bluetooth.org/en-us/Documents/Bluetooth4-2QuickRefGuide.pdf`.

[39] Weiping Sun, Munhwan Choi, and Sunghyun Choi. “IEEE 802.11ah: A Long Range 802.11 WLAN at Sub 1 GHz”. In: *Journal of ICT Standardization* 1 (2013), pp. 83–108.

Appendix A Gateway HTTP interface

Table A.1: Resource definitions

Method	URI	Resource description
GET	/sensors	
GET	/sensors/ble	Returns list of Bluetooth LE sensors
GET	/sensors/ble/{uuid}	Returns specific Bluetooth LE sensor
PUT	/sensors/ble/{uuid}	Reconfigure the sensor (i.e., configuration of threshold, scale and sample rate)
DELETE	/sensors/ble/{uuid}	Delete sensor permanently. Gateway will not add the sensor again automatically even if it is scanned again
GET	/sensors/ble/{uuid}/data	Request the newest cached data from the gateway
GET	/sensors/ble/{uuid}/analysis	The analysis responds with data concerning the wear plate condition
GET	/sensors/cloud	Returns list of cloud virtual sensors
GET	/sensors/cloud/{uuid}	Returns specific virtual sensor
GET	/sensors/combined	Combines representations Bluetooth Low Energy and Cloud sensors. This resource identifier returns list of all available combined sensors.
GET	/sensors/combined/{uuid}	Returns specific combined sensor representation

Appendix B Bluetooth 4.0 Core Architecture

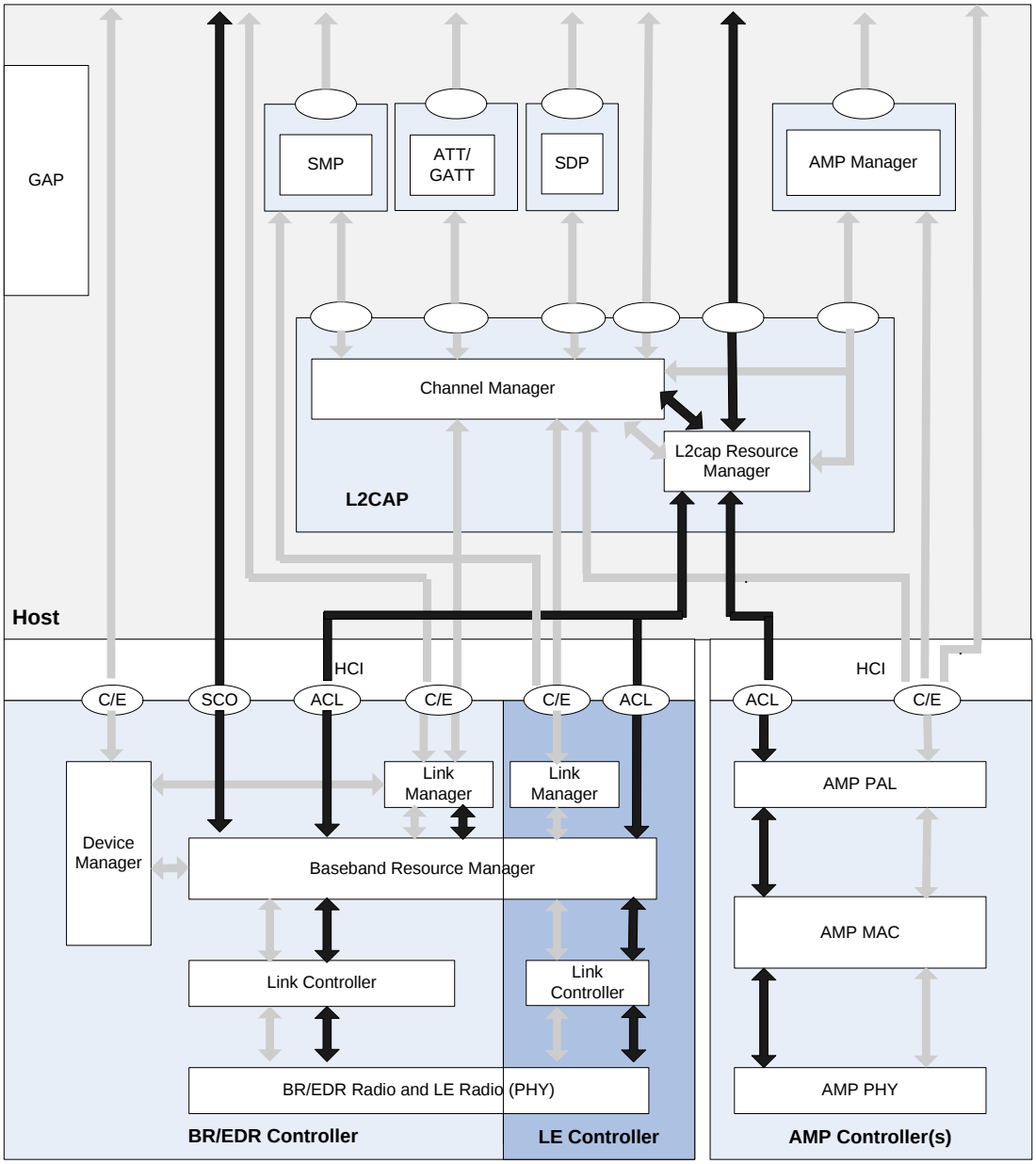

Figure B.1: Bluetooth 4.0 Core Architecture. [7]

Appendix C Gateway Web UI

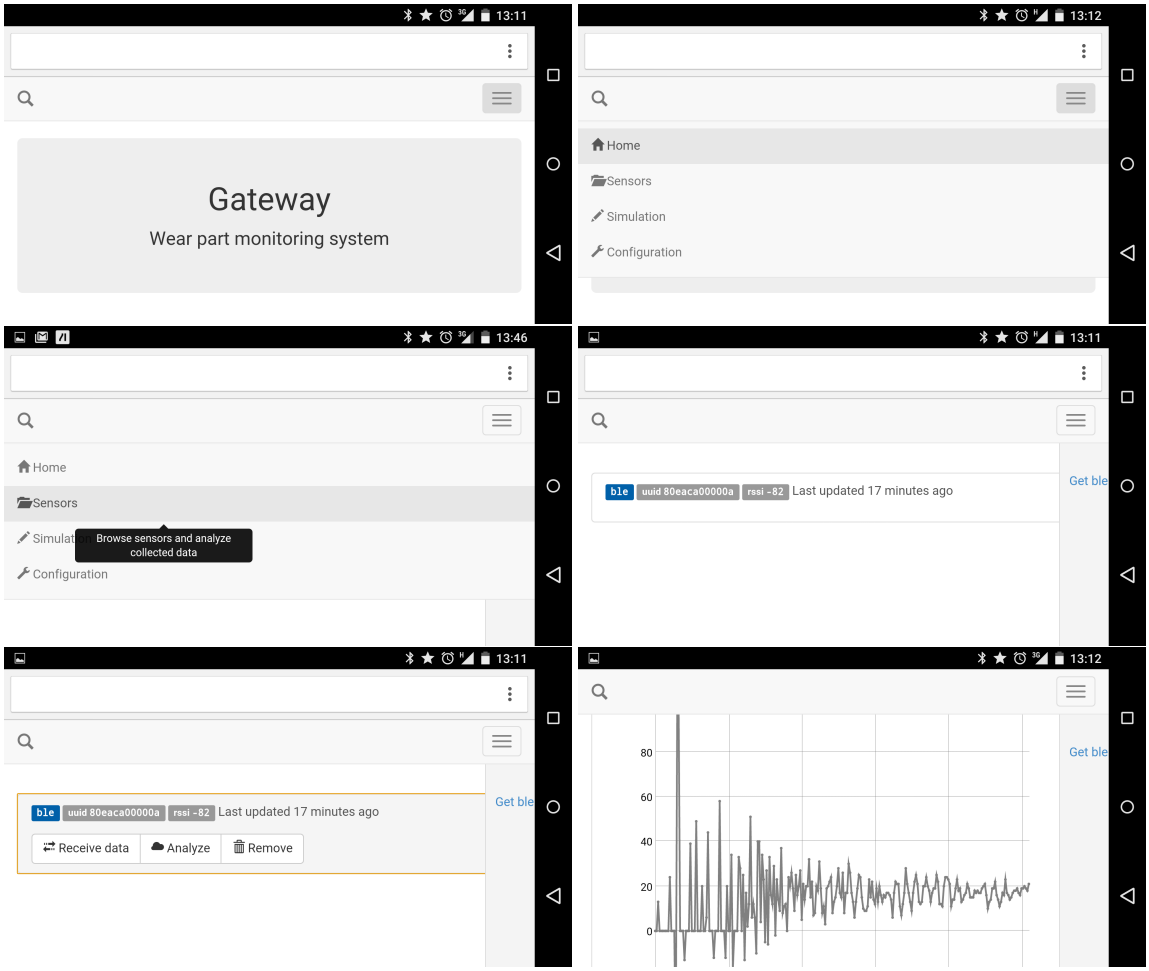

Figure C.1: Gateway shares Web UI on its HTTP server. This picture collection shows features of the mobile friendly Web UI